Joint Computation Offloading and Resource Allocation in Mobile-Edge Cloud Computing: A Two-Layer Game Approach

Zhenli He, *Senior Member, IEEE*, Ying Guo, Xiaolong Zhai, Mingxiong Zhao, *Member, IEEE*, Wei Zhou, and Keqin Li, *Fellow, IEEE*

***Abstract*—Mobile-Edge Cloud Computing (MECC) plays a crucial role in balancing low-latency services at the edge with the computational capabilities of cloud data centers (DCs). However, many existing studies focus on single-provider settings or limit their analysis to interactions between mobile devices (MDs) and edge servers (ESs), often overlooking the competition that occurs among ESs from different providers. This article introduces an innovative two-layer game framework that captures independent self-interested competition among MDs and ESs, providing a more accurate reflection of multi-vendor environments. Additionally, the framework explores the influence of cloud-edge collaboration on ES competition, offering new insights into these dynamics. The proposed model extends previous research by developing algorithms that optimize task offloading and resource allocation strategies for both MDs and ESs, ensuring the convergence to Nash equilibrium in both layers. Simulation results demonstrate the potential of the framework to improve resource efficiency and system responsiveness in multi-provider MECC environments.**

***Index Terms*—Computation offloading, game theory, mobile-edge cloud computing, nash equilibrium, resource allocation.**

I. Introduction

A. Motivation

MOBILE Edge Computing (MEC) has become a foundational pillar in modern computing, bringing computational resources closer to end-users. This proximity allows mobile devices (MDs) such as smartphones, PDAs, and wearables to overcome inherent limitations in processing power, communication bandwidth, storage, and battery life [1], [2], [3]. By leveraging MEC, MDs are now capable of delivering a wide array of advanced services, including multiplayer online gaming, high-definition image processing, and intelligent personal assistants [4]. The proliferation of MEC has not only accelerated the evolution of mobile applications but has also significantly transformed social interaction, commerce, and entertainment.

Nevertheless, as the number of MDs grows and the demand for high-computation, low-latency services escalates, the limitations of MEC are becoming more pronounced. While edge servers (ESs) are effective for handling latency-sensitive tasks, they possess significantly fewer computational resources compared to cloud data centers (DCs) [5]. High-computation tasks that demand extensive processing power are better suited for DCs, thereby preventing the overburdening of edge resources.

In response to these challenges, Mobile-Edge Cloud Computing (MECC) has emerged as a vital paradigm. MECC integrates the strengths of both ESs and DCs, ensuring that latency-sensitive tasks are handled at the edge, while computationally intensive tasks are offloaded to the cloud. This balanced approach offers the flexibility required to support increasingly complex mobile applications [6], [7], [8]. Recent advancements, such as cross-layer collaborative computation offloading [9] and hierarchical federated learning [10], further underscore the critical role of cloud-edge collaboration in achieving the necessary performance and scalability [11], [12].

As mobile computing continues to advance, future scenarios are expected to feature multiple MEC service providers and a growing number of users, creating an increasingly competitive and intricate landscape [13], [14]. Within this environment, both MDs and ESs will face distinct challenges and opportunities. For MDs, the increasing number of service providers necessitates careful selection of optimal computing platforms for task offloading to enhance computational efficiency. The limited resources at the edge intensify competition among MDs. For ESs, the challenge lies in enhancing service quality while attracting more MDs. ESs must efficiently manage their computational resources and collaborate with DCs to optimize service delivery. Effective cloud-edge collaboration ensures that these resources are fully leveraged, maximizing revenue by attracting

Received 7 November 2023; revised 10 September 2024; accepted 30 January 2025. Date of publication 3 February 2025; date of current version 10 March 2025. This work was supported in part by the National Natural Science Foundation of China under Grant 62362068 and Grant 62361056, in part by the Applied Basic Research Foundation of Yunnan Province under Grant 202201AT070156, Grant 202301AT070194, Grant 202201AT070203, and Grant 202301AT070422, in part by the Open Foundation of Yunnan Key Laboratory of Software Engineering under Grant 2023SE208, and in part by the Opening Foundation of Yunnan Key Laboratory of Smart City in Cyberspace Security under Grant 202105AG070010-ZN-10. Recommended for acceptance by P. Spirakis. *(Corresponding author: Mingxiong Zhao.)*

Zhenli He, Ying Guo, Xiaolong Zhai, Mingxiong Zhao, and Wei Zhou are with the Engineering Research Center of Cyberspace, Yunnan Key Laboratory of Software Engineering, Kunming 650091, China, and also with the School of Software, Yunnan University, Kunming 650091, China (e-mail: hezl@ynu.edu.cn; guoy@mail.ynu.edu.cn; zhaixiaolong@stu.ynu.edu.cn; mx_zhao@ynu.edu.cn; zwei@ynu.edu.cn).

Keqin Li is with the Department of Computer Science, State University of New York, New Paltz, NY 12561 USA (e-mail: lik@newpaltz.edu).

This article has supplementary downloadable material available at https://doi.org/10.1109/TCC.2025.3538090, provided by the authors.

Digital Object Identifier 10.1109/TCC.2025.3538090

more users while minimizing costs associated with underutilized resources [15], [16], [17].

Despite significant advancements in computation offloading and resource allocation strategies within MEC, particularly through the application of game theory models to explore interactions between MDs and ESs, several critical limitations remain. Much of the existing research focuses exclusively on scenarios involving single vendors or centralized resource control, assuming that all computational resources are managed by a single provider or entity. These assumptions oversimplify the complexities inherent in multi-vendor environments, where competitive dynamics among ESs controlled by different service providers play a crucial role in resource optimization. Moreover, while several two-layer or hierarchical game frameworks have been proposed, they typically address either the interactions among MDs or between MDs and ESs, without fully capturing the coexistence of inter-ES competition alongside MD interactions. These models also fail to delve into the coupling relationships between these games and their collective impact on overall system performance. Additionally, although some research has considered cloud-edge collaboration, it often overlooks how this collaboration influences the competitive dynamics among ESs, thus missing the potential impact of such collaboration on inter-ES competition. It is crucial to recognize and address these limitations to ensure the effective implementation of future MECC applications.

B. Our Contributions

To address these challenges, our research introduces an innovative framework that integrates self-interested games among MDs with those among ESs, while also considering the impact of competitive dynamics in cloud-edge collaboration on ES games. Unlike existing works, our two-layer game model explicitly accounts for the competitive nature of multi-vendor environments, where ESs and MDs operate independently and competitively. This model more accurately reflects the complexities of resource allocation and computation offloading in real-world applications. Consequently, our framework offers a comprehensive and effective solution for MECC environments dominated by multiple vendors, advancing both theoretical understanding and practical implementation.

In the first layer, MDs compete to offload their computational tasks to ESs, aiming to reduce execution latency. In the second layer, ESs engage in self-interested competition to strategically allocate resources between local processing and cloud offloading. The goal is to improve resource utilization to enhance service quality, attract a larger user base, reduce the costs of underutilized resources, and indirectly increase revenue. The primary objective of this work is to develop algorithms that determine the optimal computation offloading strategies for MDs and the optimal resource allocation and offloading strategies for ESs, achieving a balanced and self-interested equilibrium for all participants.

Our key contributions are summarized as follows:

- We introduce a comprehensive mathematical framework that models the two-layer game in the MECC environment. This framework rigorously defines the self-interested objectives of both MDs and ESs, capturing the competitive dynamics within and between these layers.
- We formally define two non-cooperative games within this two-layer structure and provide proof of the existence of Nash equilibria for each game, ensuring that the strategies adopted by MDs and ESs lead to stable outcomes.
- We develop a set of algorithms that identify the optimal strategies for both MDs in their computation offloading game and for ESs in their resource allocation and offloading game. Additionally, we propose an iterative algorithm that ensures the simultaneous convergence of both layers to a Nash equilibrium in the MECC environment.
- We conduct extensive numerical simulations and comparative experiments to validate the effectiveness and robustness of the proposed framework.

The structure of this paper is as follows: Section II reviews related work. Section III introduces mathematical models for the MECC environment. Section IV formulates the two-layer game framework and demonstrates the existence of Nash equilibria. Section V presents the algorithms for optimal player responses. Section VI includes numerical examples, performance data, and comparative experiments. Section VII concludes the paper and discusses future research directions.

II. Related Work

In this section, we categorize the existing research into three primary areas based on focus and methodology: *Resource Management and Offloading Strategies in Single-Provider Environments*, *Optimal Resource Allocation and Task Offloading Using Hierarchical Game Models*, and *Cloud-Edge Collaboration and Competition in Multi-Provider Environments*. This categorization facilitates a systematic review of the literature and underscores the unique contributions of our work in contrast to previous studies. For a more comprehensive understanding of the current research landscape, readers are encouraged to consult relevant surveys [18], [19], [20].

A. Resource Management and Offloading Strategies in Single-Provider Environments

In single-provider environments, considerable research has focused on optimizing resource management and computation offloading strategies to enhance the performance and efficiency of MEC systems. These studies often employ game-theoretic approaches to address the complex challenges of resource allocation, where users compete for limited edge resources managed by a centralized provider.

For example, Zhou et al. [21] developed a game-theoretic strategy for partial computation offloading in multi-user MEC environments, effectively reducing system latency and energy consumption. Similarly, Li [22] proposed a non-cooperative game framework to stabilize MEC environments by minimizing individual costs and ensuring efficient resource utilization when multiple MDs contend for ES resources.

Building on these foundations, Wang et al. [23] introduced a Stackelberg game model in MEC systems with Non-Orthogonal

Multiple Access (NOMA), optimizing resource allocation by modeling the leader-follower dynamics between users and the ES. Guo et al. [24] explored energy harvesting in MEC systems, proposing a game-theoretic method that balances task delay reduction with sustainable energy use.

In specialized network architectures, Messous et al. [25] applied a game-theoretic framework to optimize computation offloading in Unmanned Aerial Vehicle (UAV) networks, achieving reductions in latency and energy consumption. Chu et al. [26] addressed multi-channel resource allocation in MEC with a distributed game-theoretic algorithm that improves response times and reduces energy consumption among competing MDs.

Beyond game-theoretic methods, Liu et al. [27] introduced a service mechanism for optimizing profits in centralized computing environments, offering insights into resource allocation and pricing models. Xiao et al. [28] developed CASpMV, an accelerated framework for Sparse Matrix-Vector Multiplication on the Sunway TaihuLight supercomputer, contributing to computational resource optimization in centralized systems.

However, these studies predominantly assume a centralized, single-provider environment. This assumption oversimplifies the complex and competitive dynamics present in multi-provider MEC scenarios. Consequently, the competitive interactions between multiple edge servers managed by different providers, increasingly common in real-world deployments, are not adequately addressed.

B. *Optimal Resource Allocation and Task Offloading Using Hierarchical Game Models*

Hierarchical game-theoretic models have emerged as essential tools for optimizing computation offloading strategies across multiple layers of decision-making in MEC systems. These models typically involve interactions among various stakeholders, such as MDs, ESs, and DCs, each operating at different hierarchical levels to achieve optimal resource allocation and task offloading.

Xu et al. [29] proposed a two-stage algorithm for cloud-edge collaboration, where offloading decisions are made first, followed by resource allocation. Similarly, You et al. [30] introduced a three-tier model for MEC, considering interactions among MDs, ESs, and DCs to minimize latency and optimize resource utilization.

Further exploring hierarchical interactions, Lyu et al. [31] developed a multi-leader, multi-follower Stackelberg game to address resource allocation in competitive MEC environments, focusing on the hierarchical decision-making of providers and users. Ning et al. [32] expanded this concept by exploring dynamic resource allocation in UAV-enabled MEC, enhancing system performance in rapidly changing environments.

In dense network environments, Lu et al. [33] proposed an evolutionary game-theoretic approach for adaptive computation offloading. Guo et al. [34] applied game theory to NOMA in MEC, optimizing resource allocation in IoVT networks. Similarly, Liu et al. [35] introduced a cooperative game-based job offloading approach designed to meet strict deadlines in MEC environments, emphasizing the importance of cooperation among ESs to optimize resource utilization and ensure timely task completion.

While these studies have successfully applied hierarchical game models to optimize resource allocation and task offloading, they often focus on either the interactions among MDs, between MDs and ESs, or between ESs and DCs, typically assuming cooperation across these layers. However, they generally overlook the full complexity of inter-ES competition that can arise alongside MD interactions, particularly in scenarios where multiple providers operate independently without inter-layer cooperation. Addressing this gap is essential for accurately modeling and optimizing real-world MEC environments characterized by competitive multi-provider dynamics.

C. *Cloud-Edge Collaboration and Competition in Multi-Provider Environments*

In multi-provider environments, cloud-edge collaboration and competition strategies are vital for optimizing resource allocation and task offloading. These strategies address the complex interactions between providers to enhance service quality, manage resources efficiently, and maximize profits.

Sun et al. [36] proposed a game-theoretic approach for vehicular edge computing networks that optimizes resource allocation and task offloading, considering the collaborative dynamics among ESs managed by different providers. Pham et al. [37] similarly explored partial computation offloading in parked vehicle-assisted MEC environments, using game theory to ensure efficient resource allocation among multiple ESs. Additionally, Tong et al. [38] introduced a bilateral game approach for task outsourcing in multi-access edge computing, where ESs strategically outsource tasks to other servers or cloud resources to maximize utility, which is particularly relevant when multiple providers must collaborate despite operating independently.

Further highlighting cooperation, Diamanti et al. [39] developed an incentive mechanism combined with resource allocation strategies for edge-fog networks, integrating contract theory and game theory to motivate providers to improve service quality. Cui et al. [40] examined interference-aware device allocation in MEC, emphasizing strategic interference management in multi-provider environments to optimize performance. Xiao et al. [41] investigated heat-aware vehicular task offloading using a game-theoretic method, where ESs collaborate to manage heat dissipation while maintaining service quality. Cui et al. [42] presented a two-phase game-theoretical approach for demand response in NOMA-based MEC, optimizing energy consumption and resource allocation in dynamic demand scenarios, highlighting the need for adaptable strategies in multi-provider environments.

Li et al. [43] discussed energy-efficient stochastic task scheduling in heterogeneous computing systems, offering insights into optimizing scheduling and energy management in multi-provider environments. Although not directly focused on MEC or game theory, their work is relevant to multi-provider resource optimization.

Although these studies explore the dynamics of cloud-edge collaboration and competition in multi-provider environments,

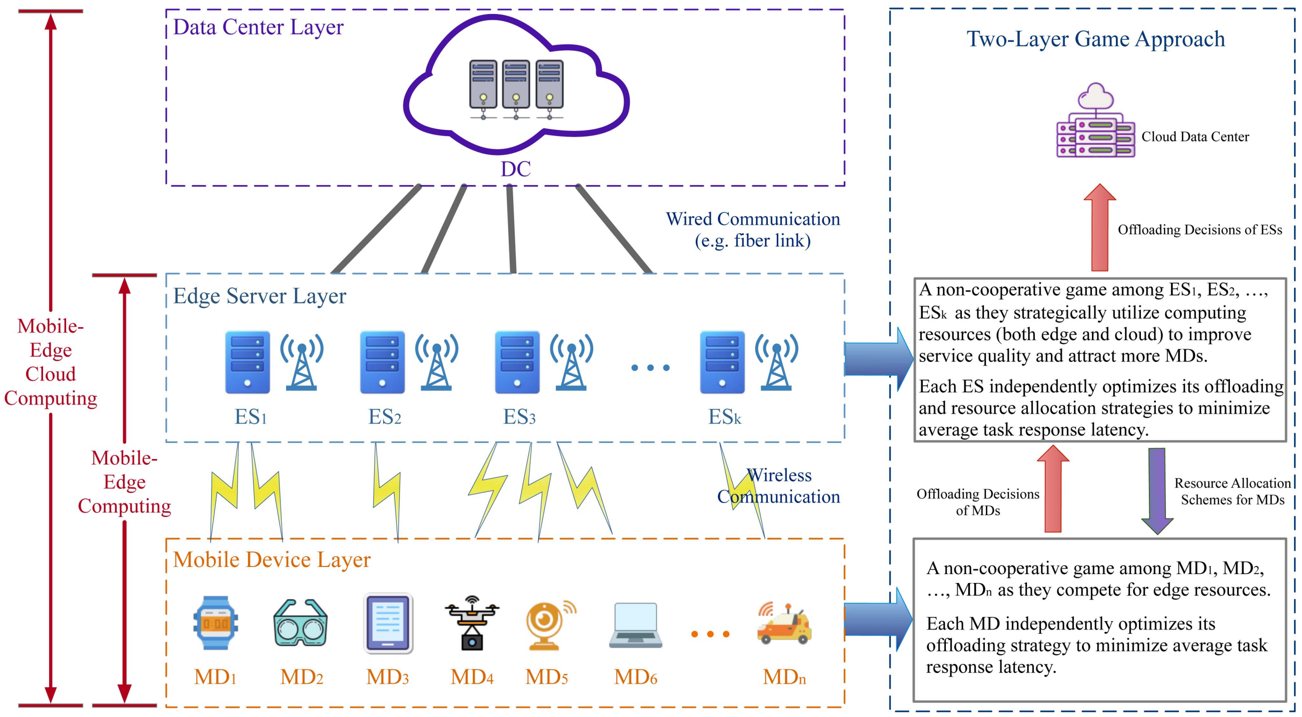
Data Center Layer
DC
Wired Communication (e.g. fiber link)
Edge Server Layer
ES1 ES2 ES3 ESk
Wireless Communication
Mobile Device Layer
MD1 MD2 MD3 MD4 MD5 MD6 MDn
Mobile-Edge Cloud Computing
Mobile-Edge Computing
Two-Layer Game Approach
Cloud Data Center
Offloading Decisions of ESs
A non-cooperative game among ES1, ES2, …, ESk as they strategically utilize computing resources (both edge and cloud) to improve service quality and attract more MDs.
Each ES independently optimizes its offloading and resource allocation strategies to minimize average task response latency.
Offloading Decisions of MDs
Resource Allocation Schemes for MDs
A non-cooperative game among MD1, MD2, …, MDn as they compete for edge resources.
Each MD independently optimizes its offloading strategy to minimize average task response latency.

Fig. 1. Overview of the MECC environment.

they often overlook how cloud-edge collaboration influences the competitive dynamics among ESs. Furthermore, while they address competition and cooperation, these studies do not fully explore how these interactions impact overall system performance.

D. Summary of Differences and Innovations

Our research addresses several critical limitations in existing studies. First, previous research on single-provider environments has not effectively tackled the competitive interactions that occur between multiple ESs managed by different providers. Second, while earlier studies on hierarchical game models have focused on competition among MDs, between MDs and ESs, or between ESs and DCs, they have often overlooked the competition among ESs from different providers, which is becoming increasingly important. Our study uniquely considers the coexistence of independent competition among both MDs and ESs, reflecting a more realistic situation where each group competes within its own dynamics without considering the other's internal competition. This approach offers a new way to achieve equilibrium in multi-provider MEC environments. Finally, while some studies have explored cloud-edge collaboration and competition in multi-provider settings, they often miss the complex dependencies and feedback loops between cloud-edge collaboration and competition among ESs. Our research examines these feedback loops, highlighting how cloud-edge collaboration influences ES competition and providing a clearer understanding of how these interactions affect overall system performance.

III. Preliminaries

In this section, we present the necessary preliminaries, including definitions, notations, and models that will be utilized throughout the paper. Due to space constraints, a detailed summary of notations and definitions is provided in Section I of the supplementary material for better readability and easy reference, available online.

A. The Mobile-Edge Cloud Computing Environment

In this section, we outline the key characteristics of the MECC environment considered in this paper.

The MECC environment comprises multiple heterogeneous computing nodes, including multiple MDs, multiple ESs, and a single DC. These nodes form a three-tier architecture, where MDs offload computational tasks to ESs, and ESs may further offload some tasks to the DC, as illustrated in Fig. 1.

- *Mobile Device Layer:* The mobile device layer consists of n competitive and self-interested MDs (denoted as $\mathrm{MD}_1, \mathrm{MD}_2, \ldots, \mathrm{MD}_n$) that compete for edge resources by offloading their computational tasks to multiple heterogeneous ESs within a geographic region, aiming to minimize the average response latency for their tasks. Each MD's decision involves selecting multiple ESs as offloading targets and determining the corresponding offloading ratios. The average response latency for each MD is influenced by both local processing and the offloading strategy.
- *Edge Server Layer:* This layer consists of k competitive and self-interested ESs (denoted as $\mathrm{ES}_1, \mathrm{ES}_2, \ldots, \mathrm{ES}_k$) that compete with each other to attract more MDs by strategically utilizing their computing and cloud resources

to enhance service quality, specifically by minimizing the average response latency of tasks executed on each ES. To achieve this, each ES must define both its offloading scheme (i.e., the proportion of tasks to be offloaded to the DC) and its resource allocation scheme (i.e., the allocation of computing resources to the MDs it serves), ensuring efficient task processing and optimized resource usage.

- *Data Center Layer:* The data center layer consists of a remote DC with theoretically limitless computational resources. Tasks offloaded to the DC are assumed to be executed within predetermined time constraints. However, due to the significant geographical distance between the DC and the ESs, transmission and propagation latencies are introduced during the task upload process. The DC's role is limited to providing flexible offloading services to the ESs, without engaging in direct competition or gaming activities, focusing exclusively on processing computation-intensive tasks offloaded by the ESs.

The interaction between MDs and ESs is driven by their respective goals: MDs seek to minimize their average response latency by optimizing offloading decisions, including selecting ESs and offloading ratios, while ESs aim to reduce task latency through resource allocation and deciding whether to offload tasks to the DC. These decisions are interdependent, as MDs' offloading choices affect ESs' task loads, and ESs' resource and offloading strategies influence the overall latency experienced by MDs. The central challenge of this paper is to balance these objectives to optimize system performance in the MECC environment.

B. The MD Model

In the MECC environment, each MD generates multiple computational tasks, each of which requires a certain number of CPU cycles to execute.

To simplify the representation, we use a tuple (r_i, c_i) to describe the computational tasks generated by MD$_i$. Here, r_i represents the input data sizes of all tasks generated by MD$_i$, measured in millions of bits (Mb), and c_i represents the number of CPU cycles required to process 1-bit data, measured in cycles/bit. Note that the number of CPU cycles required to process 1-bit data may vary for different tasks generated by different MDs.

Furthermore, taking into account the different computational capacities of each MD, we represent the computational capacity of MD$_i$ as f_i (a.k.a. the computing resource of MD$_i$), measured in MHz.

To minimize the average response latency of its own tasks, MD$_i$ may need to offload its computational tasks to multiple ESs for remote execution when its computational capacity is limited. We assume that the computational tasks generated by MDs can be divided into arbitrary segments at the bit level. Hence, tasks of MD$_i$ with an input data size of r_i can be partitioned into $k+1$ parts, denoted as $\lambda_{i,0}, \lambda_{i,1}, \lambda_{i,2}, \ldots, \lambda_{i,k}$. Here, $\lambda_{i,0}$ represents the proportion of computational tasks processed locally by MD$_i$, and $\lambda_{i,j}$ represents the proportion of computational tasks offloaded to ES$_j$, where $1 \leq j \leq k$.

Based on the given information, we can conclude that $\boldsymbol{\lambda_i} = (\lambda_{i,0}, \lambda_{i,1}, \lambda_{i,2}, \ldots, \lambda_{i,k})$ represents the offloading decision for MD$_i$, where $\lambda_{i,j} \in [0, 1]$. We have $\lambda_{i,0} + \sum_{j=1}^{k} \lambda_{i,j} = 1$ and $\lambda_{i,0} r_i + r_i \sum_{j=1}^{k} \lambda_{i,j} = r_i$.

It should be noted that the division of tasks into $k+1$ parts is dynamically determined by the offloading strategy, which optimizes the distribution of tasks across the available computational resources. MDs do not always partition their tasks across all $k+1$ units. Moreover, offloading communication takes place over distinct wireless channels between MDs and ESs, easing the load on the core network and minimizing the risk of congestion.

C. The ES Model

In the MECC environment, each ES can receive computational tasks of varying sizes offloaded from n MDs. To minimize the average response latency of these offloaded tasks and attract more MDs, each ES must decide whether to offload all or part of the received computational tasks to the DC, as well as determine the allocation scheme of its computing resources to the MDs.

As previously mentioned, the data sizes of computational tasks offloaded from MD$_i$ to ES$_j$ are $\lambda_{i,j} r_i$. Therefore, ES$_j$ will receive a set of tasks from n MDs, which can be represented as a vector $(\lambda_{1,j} r_1, \lambda_{2,j} r_2, \ldots, \lambda_{n,j} r_n)$. Let $\dot{\lambda}_{i,j}$ represent the proportion of computational tasks offloaded from MD$_i$ to ES$_j$, which will be subsequently offloaded to the DC for remote execution. Then, the vector $\dot{\boldsymbol{\lambda}}_{\boldsymbol{j}} = (\dot{\lambda}_{1,j}, \dot{\lambda}_{2,j}, \ldots, \dot{\lambda}_{n,j})$ constitutes the offloading decision of ES$_j$, where $\dot{\lambda}_{i,j} \in [0, 1]$. It is evident that the data of size $\dot{\lambda}_{i,j} \lambda_{i,j} r_i$ in the computational tasks offloaded from MD$_i$ to ES$_j$ will be sent to the DC for remote execution, while the remaining portion of the data (with a size of $(1 - \dot{\lambda}_{i,j}) \lambda_{i,j} r_i$) will stay at ES$_j$ for local execution.

We will now discuss the resource allocation scheme for ES$_j$. Let F_j represent the total computing resources available to ES$_j$ (measured in MHz). Then, the resource allocation scheme of ES$_j$ can be expressed as a vector $\boldsymbol{f_j} = (f_{1,j}, f_{2,j}, \ldots, f_{n,j})$, where $f_{i,j}$ represents the computing resources allocated by ES$_j$ to MD$_i$, and $f_{i,j} \in [0, F_j]$. The computing resources allocated by ES$_j$ for n MDs cannot exceed its upper limit. Therefore, we establish the constraint $\sum_{i=1}^{n} f_{i,j} \leq F_j$. Importantly, ES$_j$ cannot allocate computing resources to MDs that have not offloaded computational tasks onto it. Thus, when $\lambda_{i,j} = 0$, $f_{i,j} = 0$.

D. The DC Model

In theory, the DC has infinite computing resources [44], [45]. Therefore, we assume that the DC can guarantee completion of execution within t_{dc} for any input data size, implying that processing latency remains constant regardless of data size.

Furthermore, considering the typical location of the DC at the core of the network, we account for propagation and transmission latencies when offloading tasks from ESs to the DC. Here, we denote R_{dc} (measured in Mbps) as the average data transmission rate in the Wide Area Network (WAN), and d (measured in seconds) as the average propagation latency associated with data transmission from ESs to the DC.

E. Transmission Latency

In this section, we discuss the calculation of transmission latency during the task offloading process.

First, we derive the transmission latency experienced by MDs when offloading computational tasks to ESs. According to Shannon's theorem [46], the data transmission rate between MD$_i$ and ES$_j$ can be mathematically defined as

$$R_{i,j} = b_{i,j} \log_2 \left(1 + \frac{p_{i,j} h_{i,j}}{b_{i,j} N_i}\right). \tag{1}$$

Here, $b_{i,j}$ (measured in MHz) represents the communication channel bandwidth between MD$_i$ and ES$_j$, $h_{i,j}$ (measured in dBm) denotes the channel gain between MD$_i$ and ES$_j$, $p_{i,j}$ (measured in Watts) is the transmission power for MD$_i$ when transmitting data to ES$_j$, and N_i (measured in dBm/Hz) stands for the noise power spectrum density. Considering that MD$_i$ offloads $\lambda_{i,j} r_i$ Mb of tasks to ES$_j$, the transmission latency during the task offloading process can be precisely calculated as

$$t_{i,j}^{trans} = \frac{\lambda_{i,j} r_i}{R_{i,j}}. \tag{2}$$

Second, we derive the transmission latency experienced by ESs when offloading computational tasks to the DC. As previously mentioned, the DC receives computational tasks with $\dot{\lambda}_{i,j} \lambda_{i,j} r_i$ Mb from ES$_j$. The average data transmission rate for ESs communicating with the DC is denoted as R_{dc}, and the average propagation latency is represented as d. Consequently, the transmission latency during the task offloading process can be precisely calculated as

$$\dot{t}_{i,j}^{trans} = \frac{\dot{\lambda}_{i,j} \lambda_{i,j} r_i}{R_{dc}} + d. \tag{3}$$

F. Processing Latency

In this section, we analyze the processing latency for tasks executed locally on MDs, as well as the processing latency for tasks executed remotely on ESs and the DC.

First, we derive the processing latency of tasks generated on MD$_i$ and executed locally on MD$_i$, denoted by t_i^{exec}, where $1 \leq i \leq n$. According to the MD model defined in Section III-B, MD$_i$ processes computational tasks of size $\lambda_{i,0} r_i$ Mb. Therefore, the processing latency of tasks processed on MD$_i$ can be calculated as

$$t_i^{exec} = \frac{c_i \lambda_{i,0} r_i}{f_i}. \tag{4}$$

Second, we derive the processing latency of tasks that are offloaded from MD$_i$ and will be processed on ES$_j$, denoted by $t_{i,j}^{exec}$, where $1 \leq j \leq k$. Based on the ES model defined in Section III-C, ES$_j$ processes computational tasks of size $(1 - \dot{\lambda}_{i,j}) \lambda_{i,j} r_i$ Mb. Therefore, the processing latency of tasks processed on ES$_j$ can be expressed as

$$t_{i,j}^{exec} = \frac{c_i (1 - \dot{\lambda}_{i,j}) \lambda_{i,j} r_i}{f_{i,j}}. \tag{5}$$

Third, according to the DC model defined in Section III-D, the processing latency of tasks processed on the DC is t_{dc}.

IV. Two-Layer Non-Cooperative Game Framework

In this section, we formulate a two-layer non-cooperative game framework for the considered MECC environment.

A. The First Layer Game

In the first layer game, there are n non-cooperative MDs competing for edge resources.

In this non-cooperative game, we have n players designated as $\text{MD}_1, \text{MD}_2, \ldots, \text{MD}_n$. We use the notation $\boldsymbol{x_i}$ to represent the strategy of player MD$_i$ and $\boldsymbol{x_{-i}}$ to represent the strategies of the other MDs. Additionally, we define $u_i(\boldsymbol{x_i})$ as the payoff function for player MD$_i$. Therefore, the set of strategies and the payoff function of the game can be described as follows:

- Game($\boldsymbol{\lambda_i}$): $\boldsymbol{x_i} = \boldsymbol{\lambda_i}$, $\boldsymbol{\lambda_i} = (\lambda_{i,0}, \lambda_{i,1}, \ldots, \lambda_{i,k}) \in K_i$, and $u_i(\boldsymbol{x_i}, \boldsymbol{x_{-i}}) = t_i(\boldsymbol{\lambda_i})$.

In this context, the strategy set K_i is a convex set, defined as:

$$K_i = \{(\lambda_{i,0}, \lambda_{i,1}, \ldots, \lambda_{i,k}) | \lambda_{i,0} + \lambda_{i,1} + \cdots + \lambda_{i,k} = 1\}.$$

The payoff function $t_i(\boldsymbol{\lambda_i})$ for MD$_i$ represents the average response latency for all tasks generated on MD$_i$. This latency can be broken down into three key components:

- *(1) Local execution latency:* The time required to process tasks that are executed locally on the MD.
- *(2) Offloading latency to ES:* The time required to transmit tasks to one or more ESs and have them executed remotely. This latency includes both the transmission latency for sending the tasks from MD to ES and the execution latency at the ES.
- *(3) Offloading latency from ES to DC (if applicable):* For tasks further offloaded by an ES to the DC, the offloading latency also includes the transmission latency from the ES to the DC, and the remote execution latency at the DC.

In summary, the average response latency $t_i(\boldsymbol{\lambda_i})$ for MD$_i$ is a weighted sum of these components, accounting for the offloading ratios to each computing unit (MD, ES, and DC). Mathematically, this is expressed as:

$$\begin{aligned} t_i = {} & \frac{\lambda_{i,0} r_i}{r_i} t_i^{exec} + \sum_{j=1}^{k} \frac{(1 - \dot{\lambda}_{i,j}) \lambda_{i,j} r_i}{r_i} \left(t_{i,j}^{trans} + t_{i,j}^{exec}\right) \\ & + \sum_{j=1}^{k} \frac{\dot{\lambda}_{i,j} \lambda_{i,j} r_i}{r_i} \left(t_{i,j}^{trans} + \dot{t}_{i,j}^{trans} + t_{dc}\right) \\ = {} & \frac{\lambda_{i,0} r_i}{r_i} \cdot \frac{c_i \lambda_{i,0} r_i}{f_i} + \sum_{j=1}^{k} \left(\frac{(1 - \dot{\lambda}_{i,j}) \lambda_{i,j} r_i}{r_i} \right. \\ & \left. \times \left(\frac{\lambda_{i,j} r_i}{R_{i,j}} + \frac{c_i (1 - \dot{\lambda}_{i,j}) \lambda_{i,j} r_i}{f_{i.j}} \right) \right) \\ & + \sum_{j=1}^{k} \frac{\dot{\lambda}_{i,j} \lambda_{i,j} r_i}{r_i} \left(\frac{\lambda_{i,j} r_i}{R_{i,j}} + \frac{\dot{\lambda}_{i,j} \lambda_{i,j} r_i}{R_{dc}} + d + t_{dc} \right) \end{aligned}$$

$$= \frac{c_i \lambda_{i,0}^2 r_i}{f_i} + \sum_{j=1}^{k} \left(\frac{(1-\dot{\lambda}_{i,j})\lambda_{i,j}^2 r_i}{R_{i,j}} + \frac{(1-\dot{\lambda}_{i,j})^2 \lambda_{i,j}^2 r_i}{f_{i,j}} \right) + \sum_{j=1}^{k} \left(\frac{\dot{\lambda}_{i,j}\lambda_{i,j}^2 r_i}{R_{i,j}} + \frac{\dot{\lambda}_{i,j}^2 \lambda_{i,j}^2 r_i}{R_{dc}} + \dot{\lambda}_{i,j}\lambda_{i,j} d + \dot{\lambda}_{i,j}\lambda_{i,j} t_{dc} \right). \tag{6}$$

B. The Second Layer Game

In the second layer game, k non-cooperative ESs compete with each other to attract more MDs by strategically utilizing their computing and cloud resources to enhance the quality of service.

Let $\mathrm{ES}_1, \mathrm{ES}_2, \ldots, \mathrm{ES}_k$ be the k players in this non-cooperative game. Each ES_j manipulates two variables, $\dot{\lambda}_{i,j}$ and $f_{i,j}$, which directly impact the average response latency of tasks executed on ES_j. These variables are treated as Subgame($\dot{\boldsymbol{\lambda}}_{\boldsymbol{j}}$) and Subgame($\boldsymbol{f_j}$), respectively.

- Subgame($\dot{\boldsymbol{\lambda}}_{\boldsymbol{j}}$): $\boldsymbol{x_j} = \dot{\boldsymbol{\lambda}}_{\boldsymbol{j}}, \dot{\boldsymbol{\lambda}}_{\boldsymbol{j}} = (\dot{\lambda}_{1,j}, \dot{\lambda}_{2,j}, \ldots, \dot{\lambda}_{n,j}) \in K_j'$, and $u_j(\boldsymbol{x_j}, \boldsymbol{x_{-j}}) = T_j(\dot{\boldsymbol{\lambda}}_{\boldsymbol{j}})$.
- Subgame($\boldsymbol{f_j}$): $\boldsymbol{x_j} = \boldsymbol{f_j}, \boldsymbol{f_j} = (f_{1,j}, f_{2,j}, \ldots, f_{n,j}) \in K_j$, and $u_j(\boldsymbol{x_j}, \boldsymbol{x_{-j}}) = T_j(\boldsymbol{f_j})$.

Since $\dot{\lambda}_{i,j}$ and $f_{i,j}$ jointly impact the average response latency of tasks executed on ES_j, we define Game($\dot{\boldsymbol{\lambda}}_{\boldsymbol{j}}, \boldsymbol{f_j}$). Here, $\boldsymbol{x_j}$ represents the strategy of player ES_j, and $\boldsymbol{x_{-j}}$ refers to the strategies of the other ESs. The payoff function $u_j(\boldsymbol{x_j})$ denotes the performance of ES_j. The set of strategies and the payoff function for this game are as follows:

- Game($\dot{\boldsymbol{\lambda}}_{\boldsymbol{j}}, \boldsymbol{f_j}$): $\boldsymbol{x_j} = (\dot{\boldsymbol{\lambda}}_{\boldsymbol{j}}, \boldsymbol{f_j})$, $\boldsymbol{f_j} \times \dot{\boldsymbol{\lambda}}_{\boldsymbol{j}} \in K_j \times K_j'$, and $u_j(\boldsymbol{x_j}, \boldsymbol{x_{-j}}) = T_j(\dot{\boldsymbol{\lambda}}_{\boldsymbol{j}}, \boldsymbol{f_j})$.

The strategy sets K_j and K_j' are two convex sets for all $1 \le j \le k$, defined as follows:

$$K_j = \{(f_{1,j}, f_{2,j}, \ldots, f_{n,j}) | f_{1,j} + f_{2,j} + \cdots + f_{n,j} \le F_j\},$$
$$K_j' = \left\{(\dot{\lambda}_{1,j}, \dot{\lambda}_{2,j}, \ldots, \dot{\lambda}_{n,j}) | \dot{\lambda}_{i,j} \le 1, 1 \le i \le n\right\}.$$

The payoff function $T_j(\dot{\boldsymbol{\lambda}}_{\boldsymbol{j}}, \boldsymbol{f_j})$ represents the average response latency for all tasks executed on ES_j. This latency can be divided into two main components:

- *(1) Local execution latency:* The time required for the ES to process tasks locally that are offloaded by MDs.
- *(2) Offloading latency to DC (if applicable):* For tasks further offloaded to the DC, the ES also considers the transmission latency to the DC and the remote execution latency at the DC.

The objective of ES_j is to minimize the weighted sum of these latencies, considering the effects of resource allocation and offloading decisions. According to the previous discussion, the processing latency of all tasks executed on ES_j can be calculated as:

$$\sum_{i=1}^{n} \frac{(1-\dot{\lambda}_{i,j})\lambda_{i,j} r_i}{\sum_{l=1}^{n} (1-\dot{\lambda}_{l,j})\lambda_{l,j} r_l} \left(t_{i,j}^{trans} + t_{i,j}^{exec}\right).$$

For tasks offloaded from ES_j to the DC and executed at the DC, the processing latency is:

$$\sum_{i=1}^{n} \frac{\dot{\lambda}_{i,j}\lambda_{i,j} r_i}{\sum_{l=1}^{n} \dot{\lambda}_{l,j}\lambda_{l,j} r_l} \left(t_{i,j}^{trans} + t_{i,j}^{trans} + t_{dc}\right).$$

Thus, $T_j(\dot{\boldsymbol{\lambda}}_{\boldsymbol{j}}, \boldsymbol{f_j})$ can be expressed as:

$$\begin{aligned} T_j &= \frac{\sum_{i=1}^{n} (1-\dot{\lambda}_{i,j})\lambda_{i,j} r_i}{\sum_{i=1}^{n} \lambda_{i,j} r_i} \left(\sum_{i=1}^{n} \frac{(1-\dot{\lambda}_{i,j})\lambda_{i,j} r_i}{\sum_{l=1}^{n} (1-\dot{\lambda}_{l,j})\lambda_{l,j} r_l} \times \left(\frac{\lambda_{i,j} r_i}{R_{i.j}} + \frac{c_i (1-\dot{\lambda}_{i,j})\lambda_{i,j} r_i}{f_{i,j}} \right) \right) \\ &\quad + \frac{\sum_{i=1}^{n} \dot{\lambda}_{i,j}\lambda_{i,j} r_i}{\sum_{i=1}^{n} \lambda_{i,j} r_i} \left(\sum_{i=1}^{n} \frac{\dot{\lambda}_{i,j}\lambda_{i,j} r_i}{\sum_{l=1}^{n} \dot{\lambda}_{l,j}\lambda_{l,j} r_l} \times \left(\frac{\lambda_{i,j} r_i}{R_{i.j}} + \frac{\dot{\lambda}_{i,j}\lambda_{i,j} r_i}{R_{dc}} + d + t_{dc} \right) \right) \\ &= \frac{\sum_{i=1}^{n} \left(\frac{\lambda_{i,j}{}^2 r_i{}^2}{R_{i.j}} + \frac{c_i \left(1-\dot{\lambda}_{i,j}\right)^2 \lambda_{i,j}{}^2 r_i{}^2}{f_{i,j}} \right)}{\sum_{i=1}^{n} \lambda_{i,j} r_i} \\ &\quad + \frac{\sum_{i=1}^{n} \left(\frac{\dot{\lambda}_{i,j}^2 \lambda_{i,j}{}^2 r_i{}^2}{R_{dc}} + \dot{\lambda}_{i,j}\lambda_{i,j} r_i d + \dot{\lambda}_{i,j}\lambda_{i,j} r_i t_{dc} \right)}{\sum_{i=1}^{n} \lambda_{i,j} r_i}. \end{aligned} \tag{7}$$

C. Existence of the Nash Equilibrium

In this section, we establish the existence of Nash equilibrium in the aforementioned games by demonstrating that the Hessian matrices of the above four payoff functions $t_i(\boldsymbol{\lambda_i}), T_j(\boldsymbol{f_j}), T_j(\dot{\boldsymbol{\lambda}}\boldsymbol{j})$, and $T_j(\dot{\boldsymbol{\lambda}}\boldsymbol{j}, \boldsymbol{f_j})$ are positive semidefinite, based on two key theorems in game theory. This ensures the convexity of the functions and confirms the existence of a Nash equilibrium in these games. Due to space constraints, detailed derivations and proofs are provided in Section II of the supplementary material, available online.

V. Solutions for the Nash Equilibrium

To find the Nash equilibrium, we propose an iterative algorithm composed of a series of numerical methods designed to compute the optimal responses for both MDs and ESs in their respective games.

A. Optimal Response for MDs

This section presents an algorithm for computing the optimal response for MDs. In the game setup for each MD_i (Game($\boldsymbol{\lambda_i}$)), the goal is to determine the offloading decision $\boldsymbol{\lambda_i} = (\lambda_{i,0}, \lambda_{i,1}, \ldots, \lambda_{i,k})$, such that the average response latency t_i is minimized, subject to the constraint $\lambda_{i,0} + \lambda_{i,1} + \cdots + \lambda_{i,k} = 1$. This is a convex optimization problem, which can be solved using the Lagrange multiplier method [47].

First, the constraint $\lambda_{i,0} + \lambda_{i,1} + \cdots + \lambda_{i,k} = 1$ can be represented as a function $h(\lambda_{i,0}, \lambda_{i,1}, \ldots, \lambda_{i,k}) = \lambda_{i,0} + \lambda_{i,1} +$

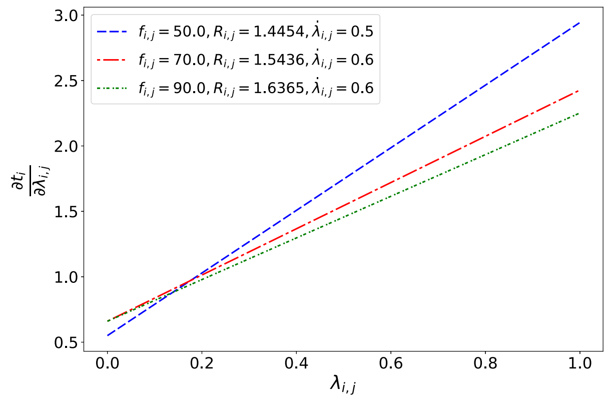

Fig. 2. Image of $\partial t_i/\partial \lambda_{i,j}$ varying with $\lambda_{i,j}$.

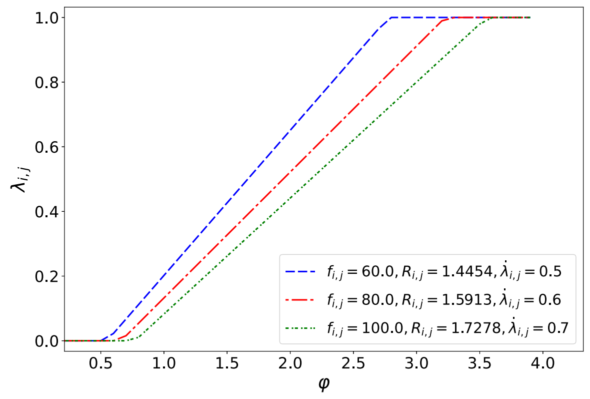

Fig. 3. Image of $\lambda_{i,j}$ varying with φ_1.

$\cdots + \lambda_{i,k} - 1$. The corresponding Lagrange function is then constructed as:

$$L = t_i(\lambda_{i,0}, \lambda_{i,1}, \ldots, \lambda_{i,k}) + \varphi_1 h\left(\lambda_{i,0}, \lambda_{i,1}, \ldots, \lambda_{i,k}\right), \quad (8)$$

leading to the necessary condition for optimally:

$$\frac{\partial t_i}{\partial \lambda_{i,j}} + \varphi_1 \frac{\partial h}{\partial \lambda_{i,j}} = 0, \quad (9)$$

where φ_1 is a Lagrange multiplier.

Next, as illustrated in Fig. 2, when the value of φ_1 is fixed, $\partial t_i/\partial \lambda_{i,j}$ is an increasing function of $\lambda_{i,j}$. We utilize this property to design Algorithm 1, which applies the classical bisection method to find $\lambda_{i,j}$ in the search interval $[S_{\min} = 0, S_{\max} = 1]$ such that $\partial t_i/\partial \lambda_{i,j} = \varphi_1$ [48, p. 22]. The algorithm terminates when the interval is shorter than ϵ (set to 10^{-12} in this paper).

By following the steps above, we obtain $\lambda_{i,j}$ that satisfies (9). We now need to check if the constraint $\lambda_{i,0} + \lambda_{i,1} + \lambda_{i,2} + \cdots + \lambda_{i,k} = 1$ holds. As shown in Fig. 3, $\lambda_{i,j}$ is an increasing function of φ_1, which implies that $\lambda_{i,0} + \lambda_{i,1} + \cdots + \lambda_{i,k}$ is also an increasing function of φ_1. Thus, we employ Algorithm 2, which is also based on the bisection method, to find φ_1 and the corresponding values of $\lambda_{i,0}, \lambda_{i,1}, \ldots, \lambda_{i,k}$ such that both conditions $\partial t_i/\partial \lambda_{i,j} = \varphi_1$ and $\lambda_{i,0} + \lambda_{i,1} + \cdots + \lambda_{i,k} = 1$ are satisfied (lines 2-12).

Algorithm 1: Search $\lambda_{i,j}$.

```
Input: $\dot{\lambda}_{i,j}, b_{i,j}, P_{i,j}, h_{i,j}, f_{i,j},\ r_i, c_i, f_i, N_0, d, t_{dc}, R_{dc}$ and $\varphi_1$.
Output: $\lambda_{i,j}$.
1: Initialize the range of $\lambda_{i,j}$ to $[S_{\min}, S_{\max}]$;
2: while $S_{\max} - S_{\min} \geq \epsilon$ do
3:     $\lambda_{i,j} \leftarrow (S_{\max} + S_{\min})/2$;
4:     Calculate $\partial t_i/\partial \lambda_{i,j}$ (readers may refer to the
       supplementary material for details, as shown in (2));
5:     if $\partial t_i/\partial \lambda_{i,j} < \varphi_1$ then
6:         $S_{\min} \leftarrow (S_{\max} + S_{\min})/2$;
7:     else
8:         $S_{\max} \leftarrow (S_{\max} + S_{\min})/2$;
9:     end if
10: end while ;
11: $\lambda_{i,j} \leftarrow (S_{\max} + S_{\min})/2$;
12: return $\lambda_{i,j}$.
```

Note that in Algorithm 2, the values of $S_{\min}$ and $S_{\max}$ are initialized as 0 and ub respectively (line 1), where ub is calculated based on the following conditions:

1) if $j = 0$, referring to (9) in the main text and (1) in the supplementary material, the maximum value of ub is obtained when $\lambda_{i,0} = 1$, giving:

$$ub = \frac{2c_i r_i}{f_i}.$$

2) if $j \neq 0$, referring to (9) in the main text and (2) in the supplementary material, the maximum value of ub is reached when $\lambda_{i,j} = 1$, resulting in:

$$ub = \frac{2r_i}{R_{i.j}} + \frac{2c_i\left(1 - \dot{\lambda}_{i,j}\right)^2 r_i}{f_{i,j}} + \frac{2\dot{\lambda}_{i,j}^2 r_i}{R_{dc}} + \dot{\lambda}_{i,j} d + \dot{\lambda}_{i,j} t_{dc}.$$

B. Optimal Response for ESs

In this section, we outline the algorithms designed to determine the optimal response for ESs. The goal for each ES, denoted as ES_j, is to determine both the offloading decision $\dot{\boldsymbol{\lambda}}_{\boldsymbol{j}} = (\dot{\lambda}_{1,j}, \dot{\lambda}_{2,j}, \ldots, \dot{\lambda}_{n,j})$ and the resource allocation scheme $\boldsymbol{f_j} = (f_{1,j}, f_{2,j}, \ldots, f_{n,j})$ in such a way that the average response latency T_j is minimized. This optimization problem is subject to the constraint $\sum_{i=1}^{n} f_{i,j} \leq F_j$, where F_j represents the total available resources at ES_j. Each of these subproblems can be viewed as a convex optimization problem and is solvable through the Karush-Kuhn-Tucker (KKT) conditions for all $1 \leq j \leq k$.

1) Solving for the Resource Allocation Subgame ($\boldsymbol{f_j}$): For **Subgame($\boldsymbol{f_j}$)**, we express the constraint $\sum_{i=1}^{n} f_{i,j} \leq F_j$ as a function $g(f_{1,j}, f_{2,j}, \ldots, f_{n,j}) = \sum_{i=1}^{n} f_{i,j} - F_j$. The Lagrange function is then constructed as follows:

$$L = T_j(f_{1,j}, f_{2,j}, \ldots, f_{n,j}) + \varphi_2 g(f_{1,j}, f_{2,j}, \ldots, f_{n,j}), \quad (10)$$

Algorithm 2: Search φ_1 and $\boldsymbol{\lambda_i}$.

```
Input: $\dot{\lambda}_{i,j}, b_{i,j}, P_{i,j}, h_{i,j}, f_{i,j}$, for all $1 \le j \le k$, $r_i, c_i, f_i, N_0, d, t_{dc}$, and $R_{dc}$.
Output: $\varphi_1$ and $\lambda_{i,0}, \lambda_{i,1}, \lambda_{i,2}, \dots, \lambda_{i,k}$.
1: Initialize the range of $\varphi_1$ to $[S_{\min}, S_{\max}]$;
2: while $S_{\max} - S_{\min} \ge \epsilon$ do
3:     $\varphi_1 \leftarrow (S_{\max} + S_{\min})/2$;
4:     for $j \in [0, k]$ do
5:         Search $\lambda_{i,j}$ using Algorithm 1;
6:     end for
7:     if $\lambda_{i,0} + \sum_{j=1}^{k} \lambda_{i,j} < 1$ then
8:         $S_{\min} \leftarrow (S_{\max} + S_{\min})/2$;
9:     else
10:        $S_{\max} \leftarrow (S_{\max} + S_{\min})/2$;
11:    end if
12: end while ;
13: $\varphi_1 \leftarrow (S_{\max} + S_{\min})/2$;
14: for $j \in [0, k]$ do
15:     Search $\lambda_{i,j}$ using Algorithm 1;
16: end for
17: return $\lambda_{i,0}, \lambda_{i,1}, \lambda_{i,2}, \dots, \lambda_{i,k}$ and $\varphi_1$.
```

where φ_2 is a Lagrange multiplier. Now, we have

$$\frac{\partial L}{\partial f_{i,j}} = \frac{\partial T_j}{\partial f_{i,j}} + \varphi_2 \frac{\partial g}{\partial f_{i,j}} = 0. \tag{11}$$

By applying the KKT conditions, we derive the following system:

$$\begin{cases} \frac{\partial T_j}{\partial f_{i,j}} + \varphi_2 \frac{\partial g}{\partial f_{i,j}} = 0, 1 \le j \le k, & \text{(a)} \\ \varphi_2 g(f_{1,j}, f_{2,j}, \dots, f_{n,j}) = 0, & \text{(b)} \\ g(f_{1,j}, f_{2,j}, \dots, f_{n,j}) \le 0, & \text{(c)} \\ \varphi_2 \ge 0. & \text{(d)} \end{cases} \tag{12}$$

By observing (12)(b)~(12)(d), the following relationship between φ_2 and $g(f_{1,j}, f_{2,j}, \dots, f_{n,j})$ can be obtained:

$$\begin{cases} \varphi_2 = 0, g(f_{1,j}, f_{2,j}, \dots, f_{n,j}) < 0, \\ \varphi_2 > 0, g(f_{1,j}, f_{2,j}, \dots, f_{n,j}) = 0. \end{cases} \tag{13}$$

However, if $\varphi_2 = 0$, (11) can be rewritten as

$$\frac{\partial L}{\partial f_{i,j}} = \frac{\partial T_j}{\partial f_{i,j}} = 0, \tag{14}$$

which implies that the equation becomes directly constant and unsolvable. Therefore, $\varphi_2 > 0$ and $g(f_{1,j}, f_{2,j}, \dots, f_{n,j}) = 0$ must hold. Then, (12) can be rewritten as

$$\begin{cases} \frac{\partial T_j}{\partial f_{i,j}} + \varphi_2 \frac{\partial g}{\partial f_{i,j}} = 0, 1 \le j \le k, \\ g(f_{1,j}, f_{2,j}, \dots, f_{n,j}) = 0, \\ \varphi_2 > 0. \end{cases} \tag{15}$$

Next, as shown in Fig. 4, we observe that $-(\partial T_j/\partial f_{i,j})$ is a decreasing function of $f_{i,j}$ for a given φ_2. Leveraging this observation, we can employ Algorithm 3 to find the value of $f_{i,j}$ that satisfies $-(\partial T_j/\partial f_{i,j}) = \varphi_2$ within a search interval (the $S_{\min}$ and $S_{\max}$ in line 1 are 0 and F_j respectively).

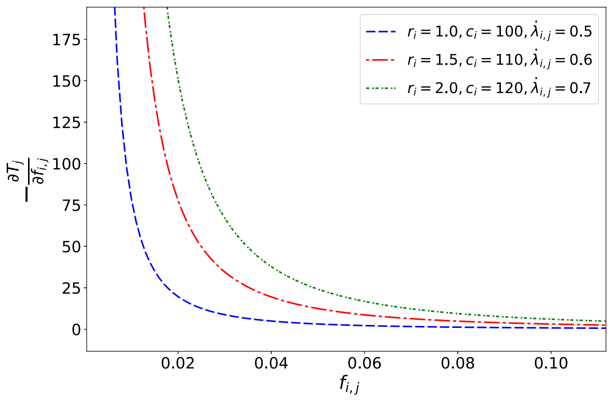

Fig. 4. Image of -$(\partial T_j/\partial f_{i,j})$ varying with $f_{i,j}$.

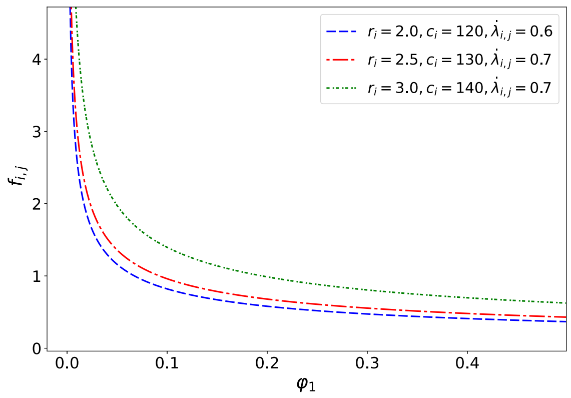

Fig. 5. Image of $f_{i,j}$ varying with φ_2.

Finally, as demonstrated in Fig. 5, we observe that $f_{i,j}$ is a decreasing function of φ_2. Consequently, the sum $f_{1,j} + f_{2,j} + \cdots + f_{n,j}$ also decreases as φ_2 increases. To determine the values of φ_2 and $f_{1,j}, f_{2,j}, \dots, f_{n,j}$, we employ the bisection method, as outlined in Algorithm 4. This ensures that $-(\partial T_j/\partial f_{i,j}) = \varphi_2$ and that $\sum_{i=1}^{n} f_{i,j} \le F_j$ are satisfied simultaneously (lines 2-12). In line 1 of Algorithm 4, the initial search range for φ_2 is $[S_{\min} = 0, S_{\max} = ub']$. As discussed previously, when $f_{i,j}$ is sufficiently small, the upper bound ub' becomes significantly large. The upper bound ub' is given by the following expression:

$$ub' = \frac{c_i \left(1 - \dot{\lambda}_{i,j}\right)^2 \lambda_{i,j}^2 {r_i}^2}{\sum_{i=1}^{n} \lambda_{i,j} r_i}.$$

2) Solving for the Offloading Decision Subgame ($\boldsymbol{\dot{\lambda}_j}$): For **Subgame($\boldsymbol{\dot{\lambda}_j}$)**, the goal for each ES$_j$ is to determine the value of $\dot{\lambda}_{i,j}$ such that $\partial T_j/\partial \dot{\lambda}_{i,j} = 0$. As demonstrated in Fig. 6, $\partial T_j/\partial \dot{\lambda}_{i,j}$ is an increasing function of $\dot{\lambda}_{i,j}$. Therefore, we propose Algorithm 5, which employs the bisection method to search for the optimal $\dot{\lambda}_{i,j}$ within the predefined interval, where the initial bounds ($S_{\min}$ and $S_{\max}$) are set to 0 and 1, respectively (line 1).

3) Iterative Algorithm for Resource Allocation and Offloading Decision: For **Game($\boldsymbol{\dot{\lambda}_j}, \boldsymbol{f_j}$)**, the offloading decisions $\boldsymbol{\dot{\lambda}_j}$ and resource allocation strategies $\boldsymbol{f_j}$ collectively

Algorithm 3: Search $f_{i,j}$.

```
Input: $\lambda_{i,j}, \dot{\lambda}_{i,j}$ $r_i$, and $c_i$, for all $1 \le i \le n$, $\varphi_2$.
Output: $f_{i,j}$.
1: Initialize the range of $f_{i,j}$ to $[S_{\min}, S_{\max}]$;
2: while $S_{\max} - S_{\min} \ge \epsilon$ do
3:     $f_{i,j} \leftarrow (S_{\max} + S_{\min})/2$;
4:     Calculate $-(\partial T_j/\partial f_{i,j})$ (readers may refer to the
       supplementary material for details, as shown in (9));
5:     if $-(\partial T_j/\partial f_{i,j}) < \varphi_2$ then
6:         $S_{\max} \leftarrow (S_{\max} + S_{\min})/2$;
7:     else
8:         $S_{\min} \leftarrow (S_{\max} + S_{\min})/2$;
9:     end if
10: end while ;
11: $f_{i,j} \leftarrow (S_{\max} + S_{\min})/2$;
12: return $f_{i,j}$.
```

Algorithm 4: Search φ_2 and $\boldsymbol{f_j}$.

```
Input: $\lambda_{i,j}, \dot{\lambda}_{i,j}, c_i$, and $r_i$, for all $1 \le i \le n$.
Output: $\varphi_2$ and $f_{1,j}, f_{2,j}, \ldots, f_{n,j}$.
1: Initialize the range of $\varphi_2$ to $[S_{\min}, S_{\max}]$;
2: while $S_{\max} - S_{\min} \ge \epsilon$ do
3:     $\varphi_2 \leftarrow (S_{\max} + S_{\min})/2$;
4:     for $i \in [1, n]$ do
5:         Search $f_{i,j}$ by using Algorithm 3;
6:     end for ;
7:     if $(f_{1,j} + f_{2,j} + \cdots + f_{n,j} < F_j)$ then
8:         $S_{\max} \leftarrow (S_{\max} + S_{\min})/2$;
9:     else
10:         $S_{\min} \leftarrow (S_{\max} + S_{\min})/2$;
11:     end if
12: end while ;
13: $\varphi_2 \leftarrow (S_{\max} + S_{\min})/2$;
14: for $i \in [1, n]$ do
15:     Search $f_{i,j}$ by using Algorithm 3;
16: end for ;
17: return $\varphi_2$ and $f_{1,j}, f_{2,j}, \ldots, f_{n,j}$.
```

determine the optimal responses for ES$_j$. We introduce an iterative algorithm (Algorithm 6) to compute these optimal responses. The algorithm begins by initializing the action profile $\boldsymbol{z} = (\boldsymbol{z_1}, \boldsymbol{z_2}, \ldots, \boldsymbol{z_k})$, where each $\boldsymbol{z_j}$ consists of the offloading decisions and resource allocation variables, $\boldsymbol{z_j} = (\dot{\lambda}_{1,j}, \dot{\lambda}_{2,j}, \ldots, \dot{\lambda}_{n,j}, f_{1,j}, f_{2,j}, \ldots, f_{n,j})$ (line 1). Each ES then uses Algorithms 3–5 to determine its optimal response, given the current state (lines 3–4).

The algorithm concludes when the difference between action profiles from two consecutive iterations is less than a predefined threshold (lines 7–12). The termination condition is:

$$\|\boldsymbol{z}' - \boldsymbol{z}\| = \sqrt{\sum_{j=1}^{k} \sum_{i=1}^{n} \left|f'_{i,j} - f_{i,j}\right|^2 + \left|\dot{\lambda}'_{i,j} - \dot{\lambda}_{i,j}\right|^2} < \delta,$$

where δ is the accuracy requirement (set to 10^{-5}). Upon convergence, the action profile $\boldsymbol{z}^* = (\boldsymbol{z_1^*}, \boldsymbol{z_2^*}, \ldots, \boldsymbol{z_k^*})$ is identified

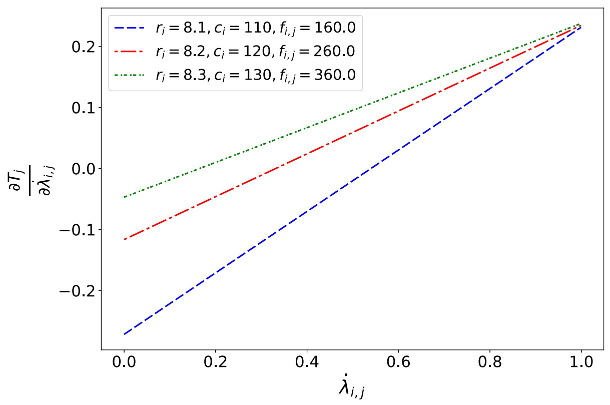

Fig. 6. Image of $\partial T_j/\partial \dot{\lambda}_{i,j}$ varying with $\dot{\lambda}_{i,j}$.

Algorithm 5: Search $\boldsymbol{\dot{\lambda}_j}$.

```
Input: $\lambda_{i,j}, c_i, r_i, f_{i,j}$, for all $1 \le i \le n$, $R_{dc}, d$, and $t_{dc}$.
Output: $\dot{\lambda}_{1,j}, \dot{\lambda}_{2,j}, \ldots, \dot{\lambda}_{n,j}$.
1: for $i \in [1, n]$ do
2:     Initialize the range of $\dot{\lambda}_{i,j}$ to $[S_{\min}, S_{\max}]$;
3:     while $S_{\max} - S_{\min} \ge \epsilon$ do
4:         $\dot{\lambda}_{i,j} \leftarrow (S_{\max} + S_{\min})/2$;
5:         Calculate $\partial T_j/\partial \dot{\lambda}_{i,j}$ (readers may refer to the
       supplementary material for details, as shown in (6));
6:         if $\partial T_j/\dot{\lambda}_{i,j} < 0$ then
7:             $S_{\min} \leftarrow (S_{\max} + S_{\min})/2$;
8:         else
9:             $S_{\max} \leftarrow (S_{\max} + S_{\min})/2$;
10:         end if
11:     end while ;
12:     $\dot{\lambda}_{i,j} \leftarrow (S_{\max} + S_{\min})/2$;
13: end for ;
14: return $\dot{\lambda}_{1,j}, \dot{\lambda}_{2,j}, \ldots, \dot{\lambda}_{n,j}$.
```

as the Nash equilibrium. At this equilibrium point, no ES can improve its outcome by unilaterally deviating from its strategy, provided that all other ESs adhere to their respective strategies.

C. An Iterative Algorithm for Nash Equilibrium

In the previous subsections, we introduced algorithms for determining the optimal responses of MDs and ESs separately. This section presents an iterative algorithm that enables both layers of non-cooperative games within the MECC environment to converge simultaneously to a Nash equilibrium.

In Algorithm 7, we begin by initializing the action profile $\mathbf{e} = (\boldsymbol{\lambda_1}, \boldsymbol{\lambda_2}, \ldots, \boldsymbol{\lambda_n}, \boldsymbol{z_1}, \boldsymbol{z_2}, \ldots, \boldsymbol{z_k})$. Each MD utilizes Algorithms 1–2 to identify its optimal response in each iteration (lines 3-5). Similarly, each ES applies Algorithm 6 to determine its optimal strategy (line 6). The algorithm proceeds iterative until the action profiles from two consecutive rounds converge within a predefined threshold (lines 7-12). The termination criterion,

Algorithm 6: Calculate the Nash Equilibrium of the Game$(\dot{\boldsymbol{\lambda}}_{\boldsymbol{j}}, \boldsymbol{f_j})$.

```
Input: $\lambda_{i,j}, \dot{\lambda}_{i,j}, f_{i,j}, r_i$ and $c_i$, for all $1 \le i \le n, 1 \le j \le k$, $R_{dc}, t_{dc}$, and $d$.
Output: The Nash equilibrium $\boldsymbol{z}^* = (\boldsymbol{z_1^*}, \boldsymbol{z_2^*}, \ldots, \boldsymbol{z_k^*})$.
1: Initialize $\boldsymbol{z} = (\boldsymbol{z_1}, \boldsymbol{z_2}, \ldots, \boldsymbol{z_k})$;
2: repeat
3: for $j \in [1, k]$ do
4:     Search $\boldsymbol{z_j}'$ by using Algorithms 3–5;
5: end for ;
6: $\boldsymbol{z}' \leftarrow (\boldsymbol{z_1}', \boldsymbol{z_2}', \ldots, \boldsymbol{z_k}')$;
7: if $\|\boldsymbol{z}' - \boldsymbol{z}\| < \delta$ then
8:     $\boldsymbol{z}^* \leftarrow \boldsymbol{z}'$;
9:     return$\boldsymbol{z}^*$;
10: else
11:     $\boldsymbol{z} \leftarrow \boldsymbol{z}'$;
12: end if
13: forever.
```

Algorithm 7: Calculate the Nash Equilibrium.

```
Input: $\lambda_{i,j}, f_{i,j}, \dot{\lambda}_{i,j}, b_{i,j}, P_{i,j}, h_{i,j}, r_i, f_i, c_i$ and $N_i$, for all $1 \le i \le n$ and $1 \le j \le k$, $R_{dc}, t_{dc}$, and $d$.
Output: The Nash equilibrium $\mathbf{e}^*=(\boldsymbol{\lambda_1}^*, \boldsymbol{\lambda_2}^*, \ldots, \boldsymbol{\lambda_n}^*, \boldsymbol{z_1^*}, \boldsymbol{z_2^*}, \ldots, \boldsymbol{z_k^*})$.
1: Initialize $\mathbf{e} = (\boldsymbol{\lambda_1}, \boldsymbol{\lambda_2}, \ldots, \boldsymbol{\lambda_n}, \boldsymbol{z_1}, \boldsymbol{z_2}, \ldots, \boldsymbol{z_k})$;
2: repeat
3: for $i \in [1, n]$ do
4:     Search $\boldsymbol{\lambda_i}'$ by using Algorithms 1–2;
5: end for ;
6: Search $\boldsymbol{z_1}', \boldsymbol{z_2}', \ldots, \boldsymbol{z_k}'$ by using Algorithm 6;
7: $\mathbf{e}' \leftarrow (\boldsymbol{\lambda_1}', \boldsymbol{\lambda_2}', \ldots, \boldsymbol{\lambda_n}', \boldsymbol{z_1}', \boldsymbol{z_2}', \ldots, \boldsymbol{z_k}')$;
8: if $\|\mathbf{e}' - \mathbf{e}\| < \delta$ then
9:     $\mathbf{e}^* \leftarrow \mathbf{e}'$;
10:     return $\mathbf{e}^*$;
11: else
12:     $\mathbf{e} \leftarrow \mathbf{e}'$;
13: end if
14: forever.
```

presented in line 8, is:

$$\|\mathrm{e}' - \mathrm{e}\| = \sqrt{\sum_{i=1}^{n} \sum_{j=0}^{k} |\lambda_{i,j}' - \lambda_{i,j}|^2 + \sum_{j=1}^{k} \sum_{i=1}^{n} (|f_{i,j}' - f_{i,j}|^2 + |\dot{\lambda}_{i,j}' - \dot{\lambda}_{i,j}|^2)} < \delta.$$

The final result is the converged action profile, denoted as $\mathbf{e}^* = (\boldsymbol{\lambda_1}^*, \boldsymbol{\lambda_2}^*, \ldots, \boldsymbol{\lambda_n}^*, \boldsymbol{z_1^*}, \boldsymbol{z_2^*}, \ldots, \boldsymbol{z_k^*})$, which corresponds to the Nash equilibrium. It is important to note that at this equilibrium, no player can gain any advantage by unilaterally deviating from e_i^*, assuming all other players remain at their equilibrium strategies.

VI. Numerical Examples and Data

In this section, we conduct extensive experiments, including comparative experiments, to analyze numerical examples and present our performance data.

A. Numerical Examples

In this section, to validate the reliability of the aforementioned series of algorithms, we conduct experiments to demonstrate and analyze the numerical results of our algorithms.

We consider a MECC environment consisting of $n = 5$ MDs, $k = 3$ ESs and a single DC. The parameters are set as follows: $r_i = 8.0 + 0.1(i-1)$ Mb, $c_i = 100 + 10(i-1)$ cycles/bit, $f_i = 80 + 20(i-1)$ MHz, $N_i = 0.1(i-1) - 174$ dBm/Hz, $B_j = 10.0 + 2(j-1)$ MHz, $R_{dc} = 50$ Mbps, $d = 1.0$ s, $t_{dc} = 0.1$ s, $\lambda_{i,j} = 0.25 + 0.005(i+j), \lambda_{i,0} = 1 - \sum_{j=1}^{k} \lambda_{i,j}$, $P_{i,j} = 7.0 + 0.5(i-1) + (j-1)$ Watts, $h_{i,j} = -32 + 0.5(i-1) + (j-1)$ dBm, $b_{i,j} = B_j/n$ MHz, $f_{i,j} = F_j/n$ MHz, $\dot{\lambda}_{i,j} = 0.05 + 0.05(i+j)$, for all $1 \le i \le n$ and $1 \le j \le k$.

Next, we present the Nash equilibrium results for computing resources $F_j = 1000 + 500(j-1)$ MHz and $F_j = 2000 + 500(j-1)$ MHz of ES$_j$ in Tables I and II, respectively. This corresponds to a scenario where all ESs have relatively limited or abundant computing resources for MDs. Specifically, we show the results of Nash equilibrium, the average response latency t_i of the tasks on MD$_i$, and the average response latency T_j of all tasks on ES$_j$. According to our numerical data, we can conclude the main observations as follows:

- For a mobile device, i.e., MD$_1$, MD$_2, \ldots,$ MD$_n$, prefer to offload computational tasks to the ES with more computing resources. Such as the fifth column $\lambda_{i,3}^*$ in Tables I and II, it is always larger than its first two columns.
- For an edge server with relatively limited computing resources, they are more willing to offload tasks to the DC for execution. As shown the tenth column $\dot{\lambda}_{i,1}^*$ in Tables I and II, it offloads more data. Due to the location of the DC is far from MDs, this has resulted in high latency and low quality of service issues, leading to a higher average response latency.

B. Performance Data

In this section, we present the performance data for the four games in detail. We consider a MECC environment with $n = 10$ MDs, $k = 5$ ESs, and a single DC. The initial parameter settings are as follows: $r_i = 10.0 + 0.1(i-1)$ Mb, $c_i = 100 + 10(i-1)$ cycles/bit, $f_i = 100 + 25(i-1)$ MHz, $N_i = 0.1(i-1) - 174$ dBm/Hz, $B_j = 15.0 + 2(j-1)$ MHz, $F_j = 1500 + 500(j-1)$ MHz, $R_{dc} = 50$ Mbps, $d = 1.0$ s, $t_{dc} = 0.1$ s, $\lambda_{i,j} = 0.01 + 0.01(i+j)$, $\lambda_{i,0} = 1 - \sum_{j=1}^{k} \lambda_{i,j}$, $P_{i,j} = 7.0 + 0.5(i+j)$ Watts, $h_{i,j} = -32 + 0.5(i+j)$ dBm, $b_{i,j} = B_j/n$ MHz, $f_{i,j} = F_j/n$ MHz, $\dot{\lambda}_{i,j} = 1.0 - 0.06(i+j)$, for all $1 \le i \le n$ and $1 \le j \le k$. For readability and comparison, the initial parameters and performance data are presented in Table III.

TABLE I
NUMERIC RESULTS FOR THE NASH EQUILIBRIUM OF $n = 5$ MDS, $k = 3$ ESS, AND A DC

i	$\lambda^*_{i,0}$	$\lambda^*_{i,1}$	$\lambda^*_{i,2}$	$\lambda^*_{i,3}$	t_i	$f^*_{i,1}$	$f^*_{i,2}$	$f^*_{i,3}$	$\dot{\lambda}^*_{i,1}$	$\dot{\lambda}^*_{i,2}$	$\dot{\lambda}^*_{i,3}$
1	0.18543	0.23982	0.27223	0.30252	1.85427	371.67783	431.26943	488.32885	0.00000	0.00000	0.00000
2	0.20401	0.23568	0.26603	0.29427	1.81775	387.87150	447.54627	504.43406	0.00000	0.00000	0.00000
3	0.21858	0.23249	0.26118	0.28774	1.85496	240.45067	430.04313	521.52364	0.40567	0.07435	0.00000
4	0.23032	0.23003	0.25725	0.28239	2.03694	3.02e-07	191.14116	365.93729	0.99999	0.60352	0.32138
5	0.23909	0.23091	0.25363	0.27637	2.14295	4.65e-09	2.74e-08	119.77615	1.00000	1.00000	0.78389
T_1=2.06458, T_2=1.98288, T_3=1.91950											

TABLE II
NUMERIC RESULTS FOR THE NASH EQUILIBRIUM OF $n = 5$ MDS, $k = 3$ ESS, AND A DC

i	$\lambda^*_{i,0}$	$\lambda^*_{i,1}$	$\lambda^*_{i,2}$	$\lambda^*_{i,3}$	t_i	$f^*_{i,1}$	$f^*_{i,2}$	$f^*_{i,3}$	$\dot{\lambda}^*_{i,1}$	$\dot{\lambda}^*_{i,2}$	$\dot{\lambda}^*_{i,3}$
1	0.18152	0.23973	0.27219	0.30654	1.81529	401.48120	467.07079	559.19780	0.00000	0.00000	0.00000
2	0.19949	0.23569	0.26611	0.29870	1.77746	419.15027	484.91178	578.63917	0.00000	0.00000	0.00000
3	0.21351	0.23259	0.26135	0.29254	1.75076	437.36356	503.56621	599.20621	0.00000	0.00000	0.00000
4	0.22475	0.23014	0.25753	0.28756	1.73221	455.92256	522.76302	620.54149	0.00000	0.00000	0.00000
5	0.23398	0.22816	0.25439	0.28345	1.77501	286.08239	521.68818	642.41532	0.39734	0.03806	0.00000
T_1=1.80294, T_2=1.76351, T_3=1.75947											

1) Game($\boldsymbol{\lambda_i}$): Through our experiments, we present the performance data for Game($\boldsymbol{\lambda_i}$) in Table IV. Compared to the initial parameters in Table III, it becomes evident that all MDs prefer to offload tasks to ESs, resulting in a marked reduction in their average response latency. For instance, the average task response latency for MD$_1$ is initially $t_1 = 5.91245$. After applying Algorithms 1 and 2, this latency is reduced to 1.86653. This demonstrates that, compared to the random offloading scheme, our proposed offloading algorithm achieves a 68.43% reduction in average response latency.

2) Subgame($\boldsymbol{f_j}$): Table V presents the performance data for Subgame($\boldsymbol{f_j}$). Compared to the initial parameters in Table III, it is clear that all ESs adjust their resource allocation schemes. For MDs with lower computation offloading demands, fewer resources are allocated, whereas MDs with higher demands receive more resources. This optimal utilization of computational resources leads to a substantial decrease in average response latency. For example, the average response latency for ES$_1$ is $T_1 = 1.44425$ under the initial resource allocation scheme. Following the application of Algorithms 3 and 4, T_1 is reduced to 1.32665. This indicates that, compared to the random resource allocation scheme, our proposed algorithm achieves an 8.14% reduction in average response latency.

3) Subgame($\boldsymbol{\dot{\lambda}_j}$): Table VI details the performance data for Subgame($\boldsymbol{\dot{\lambda}_j}$). Compared to the initial parameters in Table III, it is evident that all ESs decrease their computation offloading, leading to a noticeable reduction in average response latency. For instance, the average response latency for ES$_2$ is initially $T_2 = 1.40313$. After applying Algorithm 5, T_2 decreases to 1.23530. This finding reveals that, compared to the random offloading scheme, our proposed offloading algorithm reduces the average response latency by 11.96%.

4) Game($\boldsymbol{\dot{\lambda}_j, f_j}$): We provide the performance data for Game($\boldsymbol{\dot{\lambda}_j, f_j}$) in Table VII. Compared to the initial parameters in Table III, it becomes apparent that all ESs optimize their resource allocation and refrain from computation offloading, leading to a significant reduction in average response latency. For example, the average task response latency for ES$_1$ is $T_1 = 1.44425$ under the initial offloading and resource allocation strategy. Following the optimization in Game($\boldsymbol{\dot{\lambda}_j, f_j}$), T_1 is reduced to 1.25125. This indicates that, compared to the random offloading and resource allocation schemes, our proposed joint computation offloading and resource allocation algorithm achieves a 13.36% reduction in average response latency.

5) Numerical Results of the Nash Equilibrium: Table VIII presents the numerical results of the Nash equilibrium in the MECC environment discussed in this section. Compared to the initial parameters in Table III, it is evident that all MDs optimize their computation offloading strategies, while all ESs fine-tune both their resource allocation and computation offloading strategies. MDs are inclined to offload computational tasks to ESs with superior computing resources, thereby minimizing their average task response latency, while ESs strategically allocate their resources to enhance service quality and attract a greater number of MDs. The attainment of Nash equilibrium indicates that neither the MDs nor the ESs can further improve their payoffs by unilaterally changing their strategies. At this equilibrium point, the objectives of all MDs and ESs are fully optimized, resulting in a significant reduction in average task response latency. For instance, the average task response latency for MD$_1$ is initially $t_1 = 5.91245$. After applying the proposed algorithm, this latency is reduced to 1.49754, representing a 74.67% reduction compared to the random offloading scheme.

C. Comparison Experiments

A comparative analysis with other algorithms, including the Random Offloading Scheme (ROS), Particle Swarm Optimization (PSO), Beetle Antennae Search (BAS), and Multi-agent Proximal Policy Optimization (MAPPO) algorithms, was conducted to further demonstrate the effectiveness of our proposed solution.

TABLE III
PARAMETERS OF A MECC ENVIRONMENT WITH $n = 10$ MDs, $k = 5$ ESs, AND A SINGLE DC

	MD$_1$	MD$_2$	MD$_3$	MD$_4$	MD$_5$	MD$_6$	MD$_7$	MD$_8$	MD$_9$	MD$_{10}$
r_i	10.00000	10.10000	10.20000	10.30000	10.40000	10.50000	10.60000	10.70000	10.80000	10.90000
c_i	100.00000	110.00000	120.00000	130.00000	140.00000	150.00000	160.00000	170.00000	180.00000	190.00000
f_i	100.00000	125.00000	150.00000	175.00000	200.00000	225.00000	250.00000	275.00000	300.00000	325.00000
N_i	-174.0000	-173.9000	-173.8000	-173.7000	-173.6000	-173.5000	-173.4000	-173.3000	-173.2000	-173.1000
$P_{i,1}$	8.00000	8.50000	9.00000	9.50000	10.00000	10.50000	11.00000	11.50000	12.00000	12.50000
$P_{i,2}$	8.50000	9.00000	9.50000	10.00000	10.50000	11.00000	11.50000	12.00000	12.50000	13.00000
$P_{i,3}$	9.00000	9.50000	10.00000	10.50000	11.00000	11.50000	12.00000	12.50000	13.00000	13.50000
$P_{i,4}$	9.50000	10.00000	10.50000	11.00000	11.50000	12.00000	12.50000	13.00000	13.50000	14.00000
$P_{i,5}$	10.00000	10.50000	11.00000	11.50000	12.00000	12.50000	13.00000	13.50000	14.00000	14.50000
$h_{i,1}$	-31.00000	-30.50000	-30.00000	-29.50000	-29.00000	-28.50000	-28.00000	-27.50000	-27.00000	-26.50000
$h_{i,2}$	-30.50000	-30.00000	-29.50000	-29.00000	-28.50000	-28.00000	-27.50000	-27.00000	-26.50000	-26.00000
$h_{i,3}$	-30.00000	-29.50000	-29.00000	-28.50000	-28.00000	-27.50000	-27.00000	-26.50000	-26.00000	-25.50000
$h_{i,4}$	-29.50000	-29.00000	-28.50000	-28.00000	-27.50000	-27.00000	-26.50000	-26.00000	-25.50000	-25.00000
$h_{i,5}$	-29.00000	-28.50000	-28.00000	-27.50000	-27.00000	-26.50000	-26.00000	-25.50000	-25.00000	-24.50000
$b_{i,1}$	1.50000	1.50000	1.50000	1.50000	1.50000	1.50000	1.50000	1.50000	1.50000	1.50000
$b_{i,2}$	1.70000	1.70000	1.70000	1.70000	1.70000	1.70000	1.70000	1.70000	1.70000	1.70000
$b_{i,3}$	1.90000	1.90000	1.90000	1.90000	1.90000	1.90000	1.90000	1.90000	1.90000	1.90000
$b_{i,4}$	2.10000	2.10000	2.10000	2.10000	2.10000	2.10000	2.10000	2.10000	2.10000	2.10000
$b_{i,5}$	2.30000	2.30000	2.30000	2.30000	2.30000	2.30000	2.30000	2.30000	2.30000	2.30000
$R_{i,1}$	1.44542	1.49335	1.53825	1.58030	1.61962	1.65635	1.69061	1.72250	1.75210	1.77951
$R_{i,2}$	1.54359	1.59131	1.63604	1.67793	1.71710	1.75366	1.78772	1.81938	1.84870	1.87578
$R_{i,3}$	1.63650	1.68361	1.72776	1.76909	1.80771	1.84371	1.87719	1.90822	1.93690	1.96327
$R_{i,4}$	1.72467	1.77085	1.81411	1.85456	1.89230	1.92741	1.95999	1.99009	2.01780	2.04317
$R_{i,5}$	1.80847	1.85346	1.89556	1.93486	1.97145	2.00541	2.03682	2.06573	2.09221	2.11631
$\lambda_{i,0}$	0.75000	0.70000	0.65000	0.60000	0.55000	0.50000	0.45000	0.40000	0.35000	0.30000
$\lambda_{i,1}$	0.03000	0.04000	0.05000	0.06000	0.07000	0.08000	0.09000	0.10000	0.11000	0.12000
$\lambda_{i,2}$	0.04000	0.05000	0.06000	0.07000	0.08000	0.09000	0.10000	0.11000	0.12000	0.13000
$\lambda_{i,3}$	0.05000	0.06000	0.07000	0.08000	0.09000	0.10000	0.11000	0.1200	0.13000	0.14000
$\lambda_{i,4}$	0.06000	0.07000	0.08000	0.09000	0.10000	0.11000	0.12000	0.13000	0.14000	0.15000
$\lambda_{i,5}$	0.07000	0.08000	0.09000	0.10000	0.11000	0.12000	0.13000	0.1400	0.15000	0.16000
$f_{i,1}$	150.00000	150.00000	150.00000	150.00000	150.00000	150.00000	150.00000	150.00000	150.00000	150.00000
$f_{i,2}$	200.00000	200.00000	200.00000	200.00000	200.00000	200.00000	200.00000	200.00000	200.00000	200.00000
$f_{i,3}$	250.00000	250.00000	250.00000	250.00000	250.00000	250.00000	250.00000	250.00000	250.00000	250.00000
$f_{i,4}$	300.00000	300.00000	300.00000	300.00000	300.00000	300.00000	300.00000	300.00000	300.00000	300.00000
$f_{i,5}$	350.00000	350.00000	350.00000	350.00000	350.00000	350.00000	350.00000	350.00000	350.00000	350.00000
$\dot{\lambda}_{i,1}$	0.88000	0.82000	0.76000	0.70000	0.64000	0.58000	0.52000	0.46000	0.40000	0.34000
$\dot{\lambda}_{i,2}$	0.82000	0.76000	0.70000	0.64000	0.58000	0.52000	0.46000	0.40000	0.34000	0.28000
$\dot{\lambda}_{i,3}$	0.76000	0.70000	0.64000	0.58000	0.52000	0.46000	0.40000	0.34000	0.28000	0.22000
$\dot{\lambda}_{i,4}$	0.70000	0.64000	0.58000	0.52000	0.46000	0.40000	0.34000	0.28000	0.22000	0.16000
$\dot{\lambda}_{i,5}$	0.64000	0.58000	0.52000	0.46000	0.40000	0.34000	0.28000	0.22000	0.16000	0.10000
t_i	5.91245	4.70089	3.85427	3.22692	2.74831	2.38186	2.10864	1.91996	1.81378	1.79286
T_1=1.44425, T_2=1.40313, T_3=1.37578, T_4=1.35605, T_5=1.34138										

Random Offloading Scheme: In this scheme, the computation offloading decisions for all MDs are made at random, while ensuring that the specified constraints are satisfied. Similarly, the resource allocation scheme and computation offloading decisions for all ESs are also randomly selected, subject to the constraints. To evaluate performance, we conducted 100 random trials and selected the best outcome for comparison.

Particle Swarm Optimization: PSO is a metaheuristic algorithm that simulates the search and iteration process of particles in the solution space to solve optimization problems. In our research, the decision sets of all MDs and ESs in the current environment are treated as particles. We set the number of iterations to $K = 500$ and the number of particles in the swarm to $N = 2000$. The social coefficient is set as $c_g = 0.5$, the cognitive coefficient as $c_p = 0.5$, and the inertia weight as $\omega = 0.5$.

Beetle Antennae Search: BAS is a bio-inspired optimization method that mimics the foraging process of beetles using their antennae to locate the global optimum. In this research, the model state is initialized as the centroid coordinates of the beetle. The distance between the antennae is set to $d_0 = 2$, the initial search step to $step = 0.005$, and the step adjustment ratio to $c = 0.99$. Additionally, the number of iterations is set to $K = 10000$.

Multi-Agent Proximal Policy Optimization: MAPPO is a multi-agent reinforcement learning approach. We formulate the joint optimization problem as a Markov Decision Process (MDP) under the constraints of computation offloading ratios and available resources. This approach comprises three main components:

1) State Space: The state space includes the offloading decisions of all MDs, the offloading decisions and resource

TABLE IV
PERFORMANCE DATA OF GAME($\boldsymbol{\lambda_i}$)

	MD$_1$	MD$_2$	MD$_3$	MD$_4$	MD$_5$	MD$_6$	MD$_7$	MD$_8$	MD$_9$	MD$_{10}$
$f_{i,1}$	150.00000	150.00000	150.00000	150.00000	150.00000	150.00000	150.00000	150.00000	150.00000	150.00000
$f_{i,2}$	200.00000	200.00000	200.00000	200.00000	200.00000	200.00000	200.00000	200.00000	200.00000	200.00000
$f_{i,3}$	250.00000	250.00000	250.00000	250.00000	250.00000	250.00000	250.00000	250.00000	250.00000	250.00000
$f_{i,4}$	300.00000	300.00000	300.00000	300.00000	300.00000	300.00000	300.00000	300.00000	300.00000	300.00000
$f_{i,5}$	350.00000	350.00000	350.00000	350.00000	350.00000	350.00000	350.00000	350.00000	350.00000	350.00000
$\dot{\lambda}_{i,1}$	0.88000	0.82000	0.76000	0.70000	0.64000	0.58000	0.52000	0.46000	0.40000	0.34000
$\dot{\lambda}_{i,2}$	0.82000	0.76000	0.70000	0.64000	0.58000	0.52000	0.46000	0.40000	0.34000	0.28000
$\dot{\lambda}_{i,3}$	0.76000	0.70000	0.64000	0.58000	0.52000	0.46000	0.40000	0.34000	0.28000	0.22000
$\dot{\lambda}_{i,4}$	0.70000	0.64000	0.58000	0.52000	0.46000	0.40000	0.34000	0.28000	0.22000	0.16000
$\dot{\lambda}_{i,5}$	0.64000	0.58000	0.52000	0.46000	0.40000	0.34000	0.28000	0.22000	0.16000	0.10000
$\lambda_{i,0}$	0.15161	0.16565	0.17713	0.18748	0.19772	0.20859	0.22060	0.23412	0.24936	0.26642
$\lambda_{i,1}$	0.14396	0.14305	0.14232	0.14140	0.14005	0.13812	0.13553	0.13230	0.12848	0.12419
$\lambda_{i,2}$	0.15721	0.15542	0.15400	0.15267	0.15119	0.14940	0.14722	0.14458	0.14146	0.13788
$\lambda_{i,3}$	0.17007	0.16733	0.16512	0.16317	0.16129	0.15932	0.15716	0.15470	0.15190	0.14871
$\lambda_{i,4}$	0.18254	0.17878	0.17569	0.17302	0.17058	0.16824	0.16587	0.16337	0.16065	0.15763
$\lambda_{i,5}$	0.19461	0.18977	0.18574	0.18226	0.17917	0.17633	0.17362	0.17093	0.16815	0.16517
t_i	1.86653	1.78965	1.73143	1.69032	1.66565	1.65721	1.66491	1.68866	1.72822	1.78310

TABLE V
PERFORMANCE DATA OF SUBGAME($\boldsymbol{f_j}$)

	MD$_1$	MD$_2$	MD$_3$	MD$_4$	MD$_5$	MD$_6$	MD$_7$	MD$_8$	MD$_9$	MD$_{10}$
$\lambda_{i,0}$	0.75000	0.70000	0.65000	0.60000	0.55000	0.50000	0.45000	0.40000	0.35000	0.30000
$\lambda_{i,1}$	0.03000	0.04000	0.05000	0.06000	0.07000	0.08000	0.09000	0.10000	0.11000	0.12000
$\lambda_{i,2}$	0.04000	0.05000	0.06000	0.07000	0.08000	0.09000	0.10000	0.11000	0.12000	0.13000
$\lambda_{i,3}$	0.05000	0.06000	0.07000	0.08000	0.09000	0.10000	0.11000	0.1200	0.13000	0.14000
$\lambda_{i,4}$	0.06000	0.07000	0.08000	0.09000	0.10000	0.11000	0.12000	0.13000	0.14000	0.15000
$\lambda_{i,5}$	0.07000	0.08000	0.09000	0.10000	0.11000	0.12000	0.13000	0.1400	0.15000	0.16000
$\dot{\lambda}_{i,1}$	0.88000	0.82000	0.76000	0.70000	0.64000	0.58000	0.52000	0.46000	0.40000	0.34000
$\dot{\lambda}_{i,2}$	0.82000	0.76000	0.70000	0.64000	0.58000	0.52000	0.46000	0.40000	0.34000	0.28000
$\dot{\lambda}_{i,3}$	0.76000	0.70000	0.64000	0.58000	0.52000	0.46000	0.40000	0.34000	0.28000	0.22000
$\dot{\lambda}_{i,4}$	0.70000	0.64000	0.58000	0.52000	0.46000	0.40000	0.34000	0.28000	0.22000	0.16000
$\dot{\lambda}_{i,5}$	0.64000	0.58000	0.52000	0.46000	0.40000	0.34000	0.28000	0.22000	0.16000	0.10000
$f_{i,1}$	11.54145	24.45166	42.98631	67.77032	99.41593	138.52632	185.69820	241.52389	306.59287	381.49301
$f_{i,2}$	24.55708	43.35541	68.59740	100.93779	141.02009	189.47985	246.94713	314.04837	391.40787	479.64897
$f_{i,3}$	41.66870	66.20929	97.77325	137.01777	184.59071	241.13343	307.28299	383.67381	470.93893	569.71108
$f_{i,4}$	62.16852	92.19689	129.66659	175.22283	229.50342	293.14123	366.76601	451.00580	546.48807	653.84060
$f_{i,5}$	85.43906	120.67379	163.65546	215.00998	275.35772	345.31550	425.49814	516.51964	618.99411	733.53655
T_1=1.32665, T_2=1.29955, T_3=1.27978, T_4=1.26439, T_5=1.25223										

TABLE VI
PERFORMANCE DATA OF SUBGAME($\boldsymbol{\dot{\lambda}_j}$)

	MD$_1$	MD$_2$	MD$_3$	MD$_4$	MD$_5$	MD$_6$	MD$_7$	MD$_8$	MD$_9$	MD$_{10}$
$\lambda_{i,0}$	0.75000	0.70000	0.65000	0.60000	0.55000	0.50000	0.45000	0.40000	0.35000	0.30000
$\lambda_{i,1}$	0.03000	0.04000	0.05000	0.06000	0.07000	0.08000	0.09000	0.10000	0.11000	0.12000
$\lambda_{i,2}$	0.04000	0.05000	0.06000	0.07000	0.08000	0.09000	0.10000	0.11000	0.12000	0.13000
$\lambda_{i,3}$	0.05000	0.06000	0.07000	0.08000	0.09000	0.10000	0.11000	0.1200	0.13000	0.14000
$\lambda_{i,4}$	0.06000	0.07000	0.08000	0.09000	0.10000	0.11000	0.12000	0.13000	0.14000	0.15000
$\lambda_{i,5}$	0.07000	0.08000	0.09000	0.10000	0.11000	0.12000	0.13000	0.1400	0.15000	0.16000
$f_{i,1}$	150.00000	150.00000	150.00000	150.00000	150.00000	150.00000	150.00000	150.00000	150.00000	150.00000
$f_{i,2}$	200.00000	200.00000	200.00000	200.00000	200.00000	200.00000	200.00000	200.00000	200.00000	200.00000
$f_{i,3}$	250.00000	250.00000	250.00000	250.00000	250.00000	250.00000	250.00000	250.00000	250.00000	250.00000
$f_{i,4}$	300.00000	300.00000	300.00000	300.00000	300.00000	300.00000	300.00000	300.00000	300.00000	300.00000
$f_{i,5}$	350.00000	350.00000	350.00000	350.00000	350.00000	350.00000	350.00000	350.00000	350.00000	350.00000
$\dot{\lambda}_{i,1}$	0.00000	0.00000	0.00000	0.00000	0.18654	0.33846	0.45105	0.53697	0.60412	0.65765
$\dot{\lambda}_{i,2}$	0.00000	0.00000	0.00000	0.00000	0.05408	0.21816	0.34284	0.43989	0.51697	0.57923
$\dot{\lambda}_{i,3}$	0.00000	0.00000	0.00000	0.00000	0.00000	0.12288	0.25500	0.35950	0.44359	0.51228
$\dot{\lambda}_{i,4}$	0.00000	0.00000	0.00000	0.00000	0.00000	0.04578	0.18242	0.29193	0.38103	0.45450
$\dot{\lambda}_{i,5}$	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.12158	0.23443	0.32712	0.40417
T_1=1.27412, T_2=1.23530, T_3=1.21225, T_4=1.20014, T_5=1.19563										

TABLE VII
PERFORMANCE DATA OF GAME($\dot{\boldsymbol{\lambda}}_{\boldsymbol{j}}, \boldsymbol{f}_{\boldsymbol{j}}$)

	MD$_1$	MD$_2$	MD$_3$	MD$_4$	MD$_5$	MD$_6$	MD$_7$	MD$_8$	MD$_9$	MD$_{10}$
$\lambda_{i,0}$	0.75000	0.70000	0.65000	0.60000	0.55000	0.50000	0.45000	0.40000	0.35000	0.30000
$\lambda_{i,1}$	0.03000	0.04000	0.05000	0.06000	0.07000	0.08000	0.09000	0.10000	0.11000	0.12000
$\lambda_{i,2}$	0.04000	0.05000	0.06000	0.07000	0.08000	0.09000	0.10000	0.11000	0.12000	0.13000
$\lambda_{i,3}$	0.05000	0.06000	0.07000	0.08000	0.09000	0.10000	0.11000	0.1200	0.13000	0.14000
$\lambda_{i,4}$	0.06000	0.07000	0.08000	0.09000	0.10000	0.11000	0.12000	0.13000	0.14000	0.15000
$\lambda_{i,5}$	0.07000	0.08000	0.09000	0.10000	0.11000	0.12000	0.13000	0.1400	0.15000	0.16000
$f_{i,1}$	70.52393	99.60771	131.33366	165.64388	202.49329	241.84670	283.67666	256.89116	479.82992	1.09e-08
$f_{i,2}$	95.42359	126.35239	159.93290	196.11134	234.84596	276.10432	319.86124	366.09743	216.46060	8.81019
$f_{i,3}$	120.42929	153.08446	188.38702	226.28774	266.74852	309.73984	355.23904	403.22888	340.20384	136.65131
$f_{i,4}$	145.85570	180.25525	217.29660	256.93518	299.13657	343.87434	391.12832	440.88340	462.06847	262.56611
$f_{i,5}$	172.33560	208.63380	247.57699	289.12515	333.24759	379.92087	429.12734	480.85402	535.09174	424.08685
$\dot{\lambda}_{i,1}$	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.21670	0.87193	1.00000
$\dot{\lambda}_{i,2}$	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.47815	0.98109
$\dot{\lambda}_{i,3}$	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.25015	0.73027
$\dot{\lambda}_{i,4}$	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.06298	0.52073
$\dot{\lambda}_{i,5}$	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.28343
T_1=1.25125, T_2=1.21087, T_3=1.18614, T_4=1.17204, T_5=1.16569										

TABLE VIII
NUMERIC RESULTS FOR THE NASH EQUILIBRIUM

	MD$_1$	MD$_2$	MD$_3$	MD$_4$	MD$_5$	MD$_6$	MD$_7$	MD$_8$	MD$_9$	MD$_{10}$
$\lambda^*_{i,0}$	0.14975	0.16659	0.18015	0.19134	0.20078	0.20878	0.21524	0.22035	0.22411	0.22687
$\lambda^*_{i,1}$	0.14755	0.14499	0.14293	0.14122	0.13978	0.13917	0.14042	0.14123	0.14141	0.14120
$\lambda^*_{i,2}$	0.15936	0.15642	0.15406	0.15211	0.15047	0.14894	0.14821	0.14890	0.14893	0.14856
$\lambda^*_{i,3}$	0.17069	0.16735	0.16466	0.16245	0.16059	0.15887	0.15677	0.15590	0.15577	0.15523
$\lambda^*_{i,4}$	0.18129	0.17752	0.17448	0.17197	0.16986	0.16791	0.16559	0.16284	0.16202	0.16130
$\lambda^*_{i,5}$	0.19135	0.18713	0.18372	0.18090	0.17852	0.17633	0.17377	0.17077	0.16776	0.16684
$f^*_{i,1}$	309.46951	322.13772	334.96398	347.86146	185.56733	2.03e-08	3.41e-09	2.06e-09	1.36e-09	1.01e-09
$f^*_{i,2}$	342.63338	356.25289	370.09985	384.06883	393.84738	153.09767	1.49e-08	3.85e-09	2.32e-09	1.52e-09
$f^*_{i,3}$	375.53893	390.01919	404.79281	419.73605	434.77221	359.57259	115.56823	1.28e-08	3.83e-09	2.56e-09
$f^*_{i,4}$	406.13431	421.26222	436.74213	452.43290	468.24552	483.73723	290.0356	41.41009	7.89e-09	3.70e-09
$f^*_{i,5}$	435.43580	451.08277	467.13465	483.43350	499.87836	515.99564	445.72854	201.31075	3.04e-08	6.48e-09
$\dot{\lambda}^*_{i,1}$	0.00000	0.00000	0.00000	0.00000	0.48565	1.00000	1.00000	1.00000	1.00000	1.00000
$\dot{\lambda}^*_{i,2}$	0.00000	0.00000	0.00000	0.00000	0.01067	0.62823	1.00000	1.00000	1.00000	1.00000
$\dot{\lambda}^*_{i,3}$	0.00000	0.00000	0.00000	0.00000	0.00000	0.20005	0.75011	1.00000	1.00000	1.00000
$\dot{\lambda}^*_{i,4}$	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.41689	0.91863	1.00000	1.00000
$\dot{\lambda}^*_{i,5}$	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.15929	0.62868	1.00000	1.00000
t_i	1.49754	1.48064	1.47001	1.46405	1.49992	1.60693	1.73680	1.84419	1.87895	1.87089
T_1=1.77022, T_2=1.72182, T_3=1.68207, T_4=1.64681, T_5=1.61573										

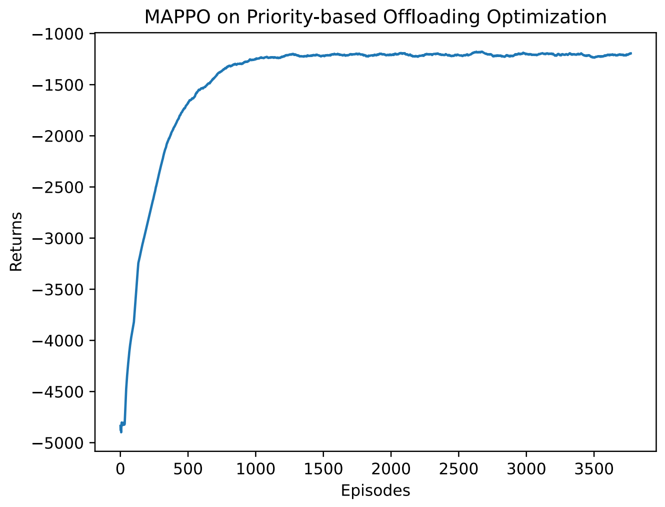

Fig. 7. MAPPO convergence performance analysis.

allocations of all ESs, the state of the DC as observed by each agent, and the rewards of other agents excluding the one in consideration within the current environment.

2) Action Space: We categorize all agents into three types, totaling 20 agents, each associated with a distinct action space. The first type of agent has a continuous action space representing the MD's computation offloading decision. The second type of agent has a discrete action space representing the ES's resource allocation scheme. The third type of agent has a continuous action space representing the ES's computation offloading decision.

3) Reward Function: The reward for each type of agent is defined as the negative value of the average response delay it incurs. For the first type of agent, the reward is defined as $R_i = -t_i$ for all $1 \le i \le n$, i.e.,

$$R_i = \begin{cases} -t_i, & \text{if } \lambda_{i,0} + \sum_{j=1}^{k} \lambda_{i,j} = 1; \\ -10^3, & \text{otherwise.} \end{cases}$$

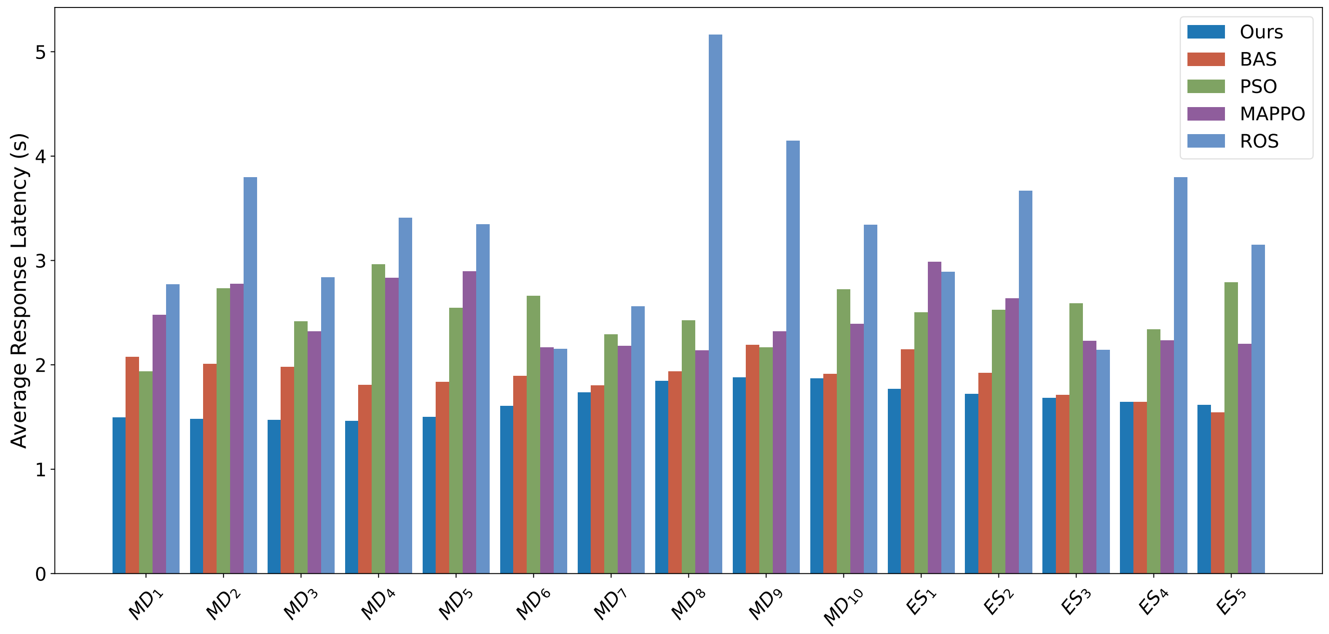

Fig. 8. Comparative analysis of average response latency across various algorithms.

TABLE IX
HYPERPARAMETERS IN MAPPO

Parameter	Value	Parameter	Value
Discount factor	0.9	GAE Lambda	0.95
Learning rate of critic	1×10^{-5}	Mini batch size	400
Learning rate of actor	5×10^{-5}	Batch size	3200
Soft-update coefficient	0.01	Entropy coefficient	0.01
Fully connected layer num	2	Value loss	MSE
Fully connected layer dim	512	Network	MLP
Learning iterations	5	Policy clip	0.2
Gradient clip norm	10	Optimizer	Adam
Max step	3.5×10^5	Activation	ReLU

For the second type of agent, the reward is defined as $R_j = -T_j$, for all $1 \leq j \leq k$, i.e.,

$$R_j = \begin{cases} -T_j, & \text{if } \sum_{i=1}^{n} f_{i,j} \leq F_j; \\ -10^3, & \text{otherwise.} \end{cases}$$

For the third type of agent, the reward function is similarly defined as $R_j = -T_j$, for all $1 \leq j \leq k$. Additionally, Table IX lists the hyperparameter settings, and Fig. 7 illustrates the convergence performance.

To maintain consistency in the comparative analysis, we use the same parameters as those listed in Table III. Fig. 8 presents the corresponding experimental results, where the x-axis denotes the MDs and ESs, and the y-axis represents their respective average response latency. The results clearly demonstrate that our proposed method outperforms the other four algorithms. Notably, the random offloading scheme exhibits the poorest performance, indicating its unsuitability for the MECC environment discussed in this paper. While the MAPPO and PSO algorithms offer some advantages, they fail to deliver optimal performance across all MDs and ESs. The novel BAS algorithm, despite its advancements, also falls short of surpassing our approach. For instance, considering MD$_5$, the average response latency is t_5 = 3.34898 using the ROS algorithm, t_5 = 2.54801 using the PSO algorithm, t_5 = 2.89755 using the MAPPO algorithm, t_5 = 1.83789 using the BAS algorithm, and t_5 = 1.49992 using our proposed algorithm. These results highlight that, in comparison with the other four algorithms, our method consistently delivers the most optimal and effective performance, achieving the best response latency for all MDs and ESs across various scenarios.

VII. Conclusions and Future Work

In this paper, we propose a novel two-layer game framework within the MECC environment to address the challenges of multi-user, multi-server computation offloading and resource allocation. Our framework thoroughly examines the intricate interactions between MDs competing for computational resources and ESs strategically allocating their resources to optimize performance.

We developed a set of algorithms that define optimal responses for MDs and ESs and presented an iterative algorithm that ensures both layers converge to a Nash equilibrium. Our experimental results demonstrate the robustness and efficacy of these algorithms, effectively improving system performance in complex, real-world, multi-vendor MECC environments. This framework expands on the current understanding of competitive interactions in MECC environments and provides practical solutions for enhancing computation offloading and resource allocation, offering a comprehensive way to manage tasks across multiple ESs while accounting for the diverse objectives of MDs and ESs. This framework offers a novel approach by integrating the competitive dynamics of MDs and ESs.

Despite these contributions, several areas remain for future exploration. Specifically, the allocation of channel bandwidth and the spatial separation between MDs and ESs were not addressed in this study, though they significantly influence task offloading decisions and could impact the performance outcomes. Furthermore, we did not account for the interactions among multiple cloud service providers, which is an emerging area of interest in MECC research. Understanding how different cloud providers compete and collaborate will be essential in future MECC frameworks. Additionally, future research will explore the incorporation of service pricing mechanisms into the resource allocation model. This enhancement will enable the exploration of how ESs can balance energy efficiency and profitability in multi-provider MECC environments, offering a more holistic framework for managing costs and attracting users.

References

[1] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things J.*, vol. 5, no. 1, pp. 450–465, Feb. 2018.

[2] A. Filali, A. Abouaomar, S. Cherkaoui, A. Kobbane, and M. Guizani, "Multi-access edge computing: A survey," *IEEE Access*, vol. 8, pp. 197017–197046, 2020.

[3] B. Liang, M. A. Gregory, and S. Li, "Multi-access edge computing fundamentals, services, enablers and challenges: A complete survey," *J. Netw. Comput. Appl.*, vol. 199, 2022, Art. no. 103308.

[4] Q.-V. Pham et al., "A survey of multi-access edge computing in 5G and beyond: Fundamentals, technology integration, and state-of-the-art," *IEEE Access*, vol. 8, pp. 116974–117017, 2020.

[5] S. Ma, S. Song, L. Yang, J. Zhao, F. Yang, and L. Zhai, "Dependent tasks offloading based on particle swarm optimization algorithm in multi-access edge computing," *Appl. Soft Comput.*, vol. 112, 2021, Art. no. 107790.

[6] J. Liu, J. Ren, Y. Zhang, X. Peng, Y. Zhang, and Y. Yang, "Efficient dependent task offloading for multiple applications in MEC-cloud system," *IEEE Trans. Mobile Comput.*, vol. 22, no. 4, pp. 2147–2162, Apr. 2023.

[7] G. Peng, H. Wu, H. Wu, and K. Wolter, "Constrained multiobjective optimization for IoT-enabled computation offloading in collaborative edge and cloud computing," *IEEE Internet Things J.*, vol. 8, no. 17, pp. 13723–13736, Sep. 2021.

[8] H. Zhou, Z. Wang, N. Cheng, D. Zeng, and P. Fan, "Stackelberg game-based computation offloading method in cloud-edge computing networks," *IEEE Internet Things J.*, vol. 9, no. 17, pp. 16510–16520, Sep. 2022.

[9] Z. He, Y. Xu, M. Zhao, W. Zhou, and K. Li, "Priority-based offloading optimization in cloud-edge collaborative computing," *IEEE Trans. Services Comput.*, vol. 16, no. 6, pp. 3906–3919, Nov./Dec. 2023.

[10] Y. Cui, K. Cao, J. Zhou, and T. Wei, "Optimizing training efficiency and cost of hierarchical federated learning in heterogeneous mobile-edge cloud computing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 5, pp. 1518–1531, May 2023.

[11] B. Shen et al., "A cloud-edge collaboration framework for generating process digital twin," *IEEE Trans. Cloud Comput.*, vol. 12, no. 2, pp. 388–404, Second Quarter, 2024.

[12] Q. Su, Q. Zhang, W. Li, and X. Zhang, "Primal-dual-based computation offloading method for energy-aware cloud-edge collaboration," *IEEE Trans. Mobile Comput.*, vol. 23, no. 2, pp. 1534–1549, Feb. 2024.

[13] M. AL-Naday, N. Thomos, J. Hu, B. Volckaert, F. de Turck, and M. J. Reed, "Service-based, multi-provider, fog ecosystem with joint optimization of request mapping and response routing," *IEEE Trans. Services Comput.*, vol. 16, no. 3, pp. 2203–2214, May/Jun. 2023.

[14] P. Charatsaris, M. Salcido, M. Diamanti, A. M. Ali, E. E. Tsiropoulou, and S. Papavassiliou, "AGORA: A multi-provider edge computing resource management and pricing framework," in *Proc. 2024 Int. Wireless Commun. Mobile Comput.*, 2024, pp. 1685–1690.

[15] Z. Chen, Q. Ma, L. Gao, and X. Chen, "Price competition in multi-server edge computing networks under SAA and SIQ models," *IEEE Trans. Mobile Comput.*, vol. 23, no. 1, pp. 754–768, Jan. 2024.

[16] Y. Peng et al., "Computing and communication cost-aware service migration enabled by transfer reinforcement learning for dynamic vehicular edge computing networks," *IEEE Trans. Mobile Comput.*, vol. 23, no. 1, pp. 257–269, Jan. 2024.

[17] N. M. Laboni, S. J. Safa, S. Sharmin, M. A. Razzaque, M. M. Rahman, and M. M. Hassan, "A hyper heuristic algorithm for efficient resource allocation in 5G mobile edge clouds," *IEEE Trans. Mobile Comput.*, vol. 23, no. 1, pp. 29–41, Jan. 2024.

[18] Z. Ning et al., "Partial computation offloading and adaptive task scheduling for 5G-enabled vehicular networks," *IEEE Trans. Mobile Comput.*, vol. 21, no. 4, pp. 1319–1333, Apr. 2022.

[19] Y. Wang et al., "A game-based computation offloading method in vehicular multiaccess edge computing networks," *IEEE Internet Things J.*, vol. 7, no. 6, pp. 4987–4996, Jun. 2020.

[20] Q. Luo, S. Hu, C. Li, G. Li, and W. Shi, "Resource scheduling in edge computing: A survey," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 4, pp. 2131–2165, Fourth Quarter, 2021.

[21] S. Zhou and W. Jadoon, "The partial computation offloading strategy based on game theory for multi-user in mobile edge computing environment," *Comput. Netw.*, vol. 178, 2020, Art. no. 107334. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128619316482

[22] K. Li, "How to stabilize a competitive mobile edge computing environment: A game theoretic approach," *IEEE Access*, vol. 7, pp. 69960–69985, 2019.

[23] K. Wang, Z. Ding, D. K. C. So, and G. K. Karagiannidis, "Stackelberg game of energy consumption and latency in MEC systems with NOMA," *IEEE Trans. Commun.*, vol. 69, no. 4, pp. 2191–2206, Apr. 2021.

[24] M. Guo, Q. Li, Z. Peng, X. Liu, and D. Cui, "Energy harvesting computation offloading game towards minimizing delay for mobile edge computing," *Comput. Netw.*, vol. 204, 2022, Art. no. 108678. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128621005491

[25] M.-A. Messous, S.-M. Senouci, H. Sedjelmaci, and S. Cherkaoui, "A game theory based efficient computation offloading in an UAV network," *IEEE Trans. Veh. Technol.*, vol. 68, no. 5, pp. 4964–4974, May 2019.

[26] S. Chu, Z. Fang, S. Song, Z. Zhang, C. Gao, and C. Xu, "Efficient multi-channel computation offloading for mobile edge computing: A game-theoretic approach," *IEEE Trans. Cloud Comput.*, vol. 10, no. 3, pp. 1738–1750, Third Quarter, 2022.

[27] C. Liu, K. Li, K. Li, and R. Buyya, "A new service mechanism for profit optimizations of a cloud provider and its users," *IEEE Trans. Cloud Comput.*, vol. 9, no. 1, pp. 14–26, First Quarter, 2021.

[28] G. Xiao, K. Li, Y. Chen, W. He, A. Y. Zomaya, and T. Li, "CASpMV: A customized and accelerative SpMV framework for the sunway TaihuLight," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 1, pp. 131–146, Jan. 2021.

[29] F. Xu, Y. Xie, Y. Sun, Z. Qin, G. Li, and Z. Zhang, "Two-stage computing offloading algorithm in cloud-edge collaborative scenarios based on game theory," *Comput. Elect. Eng.*, vol. 97, 2022, Art. no. 107624. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045790621005541

[30] F. You, W. Ni, J. Li, and A. Jamalipour, "New three-tier game-theoretic approach for computation offloading in multi-access edge computing," *IEEE Trans. Veh. Technol.*, vol. 71, no. 9, pp. 9817–9829, Sep. 2022.

[31] T. Lyu, H. Xu, and Z. Han, "Multi-leader multi-follower Stackelberg game based resource allocation in multi-access edge computing," in *Proc. 2022 IEEE Int. Conf. Commun.*, 2022, pp. 4306–4311.

[32] Z. Ning et al., "Dynamic computation offloading and server deployment for UAV-enabled multi-access edge computing," *IEEE Trans. Mobile Comput.*, vol. 22, no. 5, pp. 2628–2644, May 2023.

[33] W. Lu and X. Zhang, "Computation offloading for partitionable applications in dense networks: An evolutionary game approach," *IEEE Internet Things J.*, vol. 9, no. 21, pp. 20985–20996, Nov. 2022.

[34] F. Guo, H. Lu, B. Li, D. Li, and C. W. Chen, "NOMA-assisted multi-MEC offloading for IoVT networks," *IEEE Wireless Commun.*, vol. 28, no. 4, pp. 26–33, Aug. 2021.

[35] C. Liu, K. Li, J. Liang, and K. Li, "COOPER-MATCH: Job offloading with a cooperative game for guaranteeing strict deadlines in MEC," *IEEE Trans. Mobile Comput.*, to be published, doi: 10.1109/TMC.2019.2921713.

[36] Z. Sun, G. Sun, Y. Liu, J. Wang, and D. Cao, "BARGAIN-MATCH: A game theoretical approach for resource allocation and task offloading in vehicular edge computing networks," *IEEE Trans. Mobile Comput.*, vol. 23, no. 2, pp. 1655–1673, Feb. 2024.

[37] X.-Q. Pham, T. Huynh-The, E.-N. Huh, and D.-S. Kim, "Partial computation offloading in parked vehicle-assisted multi-access edge computing: A game-theoretic approach," *IEEE Trans. Veh. Technol.*, vol. 71, no. 9, pp. 10220–10225, Sep. 2022.

[38] Z. Tong, X. Deng, Z. Xiao, D. He, A. T. Chronopoulos, and S. Dustdar, "A bilateral game approach for task outsourcing in multi-access edge computing," *IEEE Trans. Netw. Service Manag.*, vol. 21, no. 1, pp. 266–279, Feb. 2024.

[39] M. Diamanti, P. Charatsaris, E. E. Tsiropoulou, and S. Papavassiliou, “Incentive mechanism and resource allocation for edge-fog networks driven by multi-dimensional contract and game theories,” *IEEE Open J. Commun. Soc.*, vol. 3, pp. 435–452, Feb. 2022, doi: 10.1109/OJ-COMS.2022.3154536.
[40] G. Cui, Q. He, F. Chen, Y. Zhang, H. Jin, and Y. Yang, “Interference-aware game-theoretic device allocation for mobile edge computing,” *IEEE Trans. Mobile Comput.*, vol. 21, no. 11, pp. 4001–4012, Nov. 2022.
[41] Z. Xiao et al., “Vehicular task offloading via heat-aware MEC cooperation using game-theoretic method,” *IEEE Internet Things J.*, vol. 7, no. 3, pp. 2038–2052, Mar. 2020.
[42] G. Cui et al., “Demand response in NOMA-based mobile edge computing: A two-phase game-theoretical approach,” *IEEE Trans. Mobile Comput.*, vol. 22, no. 3, pp. 1449–1463, Mar. 2023.
[43] K. Li, X. Tang, and K. Li, “Energy-efficient stochastic task scheduling on heterogeneous computing systems,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 11, pp. 2867–2876, Nov. 2014.
[44] B. Chitsaz and A. Khonsari, “General spin-up time distribution for energy-aware IaaS cloud service models,” *Cluster Comput.*, vol. 23, no. 2, pp. 1293–1301, 2020.
[45] Y. Zhang, X. Lan, J. Ren, and L. Cai, “Efficient computing resource sharing for mobile edge-cloud computing networks,” *IEEE/ACM Trans. Netw.*, vol. 28, no. 3, pp. 1227–1240, Jun. 2020.
[46] C. E. Shannon, “A mathematical theory of communication,” *Bell Syst. Tech. J.*, vol. 27, no. 3, pp. 379–423, 1948.
[47] R. T. Rockafellar, “Lagrange multipliers and optimality,” *SIAM Rev.*, vol. 35, no. 2, pp. 183–238, 1993.
[48] R. L. Burden, J. D. Faires, and A. M. Burden, *Numerical Analysis*. Boston, MA, USA: Cengage Learning, 2015.

Zhenli He (Senior Member, IEEE) received the MS degree in software engineering and the PhD degree in systems analysis and integration from Yunnan University (YNU), Kunming, China, in 2012 and 2015, respectively. He was a postdoctoral researcher with Hunan University (HNU), Changsha, China, from 2019 to 2021. He is currently an associate professor and the head with the Department of Software Engineering, School of Software, Yunnan University. His current research interests include edge computing, energy-efficient computing, heterogeneous computing, and edge AI techniques.

Ying Guo received the BE degree in computer science and technology from Yunnan University, China, in 2021. She is currently working toward the MS degree in software engineering with the School of Software, Yunnan University. Her research interests include mobile edge computing and game theory.

Xiaolong Zhai received the BS degree in software engineering from Hebei Normal University, China, in 2022. He is currently working toward the master’s degree in software engineering with Yunnan University. His research focuses on mobile edge computing and deep reinforcement learning.

Mingxiong Zhao (Member, IEEE) received the BS degree in electrical engineering and the PhD degree in information and communication engineering from the South China University of Technology (SCUT), Guangzhou, China, in 2011 and 2016, respectively. He was a visiting PhD student with the University of Minnesota (UMN), Twin Cities, MN, USA, from 2012 to 2013 and Singapore University of Technology and Design (SUTD), Singapore, from 2015 to 2016, respectively. Since 2016, he has been with the School of Software, Yunnan University, Kunming, China, where he is currently an associate professor and serves as the director of Cybersecurity Department. His current research interests include network security, mobile edge computing, and edge AI techniques.

Wei Zhou received the PhD degree from the University of Chinese Academy of Sciences. He is currently a full professor with the Software School, Yunnan University. His research interests include distributed data-intensive computing and bioinformatics. He is a fellow of the China Communications Society, a member of the Yunnan Communications Institute, and a member of the Bioinformatics Group of the Chinese Computer Society. He received the Wu Daguan Outstanding Teacher Award from Yunnan University in 2016 and was selected for the Youth Talent Program of Yunnan University in 2017. He has led several projects funded by the National Natural Science Foundation.

Keqin Li (Fellow, IEEE) received the BS degree in computer science from Tsinghua University, in 1985, and the PhD degree in computer science from the University of Houston, in 1990. He is currently a SUNY distinguished professor with the State University of New York and a national distinguished professor with Hunan University (China). He has authored or co-authored more than 950 journal articles, book chapters, and refereed conference papers. He received several best paper awards from international conferences including PDPTA-1996, NAECON-1997, IPDPS-2000, ISPA-2016, NPC-2019, ISPA-2019, and CPSCom-2022. He holds nearly 70 patents announced or authorized by the Chinese National Intellectual Property Administration. He is among the world’s top five most influential scientists in parallel and distributed computing in terms of single-year and career-long impacts based on a composite indicator of the Scopus citation database. He was a 2017 recipient of the Albert Nelson Marquis Lifetime Achievement Award for being listed in Marquis Who’s Who in Science and Engineering, Who’s Who in America, Who’s Who in the World, and Who’s Who in American Education for more than twenty consecutive years. He received the Distinguished Alumnus Award from the Computer Science Department at the University of Houston in 2018. He received the IEEE TCCLD Research Impact Award from the IEEE CS Technical Committee on Cloud Computing in 2022 and the IEEE TCSVC Research Innovation Award from the IEEE CS Technical Community on Services Computing in 2023. He is a member of the SUNY Distinguished Academy. He is an AAAS fellow, and an AAIA fellow. He is a member of Academia Europaea (Academician of the Academy of Europe).