# A Game-Based Computation Offloading With Imperfect Information in Multi-Edge Environments

Bing Lin, Jie Weng, Xing Chen, *Member, IEEE*, Yun Ma, and Ching-Hsien Hsu, *Senior Member, IEEE*

***Abstract*—Mobile Edge Computing (MEC) can augment the capability of Internet of Things (IoT) mobile devices (MDs) through offloading the computation-intensive tasks to their adjacent servers. Synergistic computation offloading among MEC servers is one possible solution to reduce the completion time of system during peak hours. However, due to the large number of servers and the long distance between base stations (BSs), synchronizing the information of all servers takes a long time, which is not applicable to the fluctuant environments. Meanwhile, each server from different BSs is typically selfish and rational, and can only obtain the imperfect information from its adjacent servers, which is a challenge for computation offloading among servers from a global perspective. This article proposes a game-based computation offloading scheme with imperfect information in multi-edge environments. First, a non-cooperative game with imperfect information is designed to analyze the complex interactions during synergistic computation offloading among MEC servers. Second, a Synergistic Balancing Offloading Algorithm (SBOA) through distributed decision-making manner to obtain the optimal offloading decision is proposed, which guarantees that the game converges to a Nash Equilibrium (NE) point. Extensive simulation results reveal the fast convergence of SBOA. As the percentage of high-load servers rises and the number of heavy tasks increases, SBOA performs better than other benchmark algorithms in terms of timeliness, effectiveness, and system completion time.**

***Index Terms*—Internet of Things (IoT), mobile edge computing (MEC), computation offloading, non-cooperative game, imperfect information.**

Received 18 March 2024; revised 4 December 2024; accepted 5 December 2024. Date of publication 16 December 2024; date of current version 6 February 2025. This work was supported in part by the National Natural Science Foundation of China under Grant 62072108, in part by the Special Funds for Promoting High-Quality Development of Marine and Fishery Industries in Fujian Province under Grant FJHYF-ZH-2023-02, in part by the Fujian Key Technological Innovation and Industrialization Projects under Grant 2024XQ004, in part by the National Key Laboratory of Data Space Technology and System under Grant QZQC2024015, and in part by the University-Industry Cooperation of Fujian Province under Grant 2022H6024. *(Corresponding author: Xing Chen.)*

Bing Lin is with the College of Physics and Energy, Fujian Provincial Key Laboratory of Quantum Manipulation and New Energy Materials, Fujian Normal University, Fuzhou 350117, China, also with the Fujian Provincial Collaborative Innovation Center for Advanced High-Field Superconducting Materials and Engineering, Fuzhou 350117, China, and also with the School of Computer Science, Peking University, Beijing 100871, China (e-mail: WheelLX@163.com).

Jie Weng and Xing Chen are with the College of Computer and Data Science, Fuzhou University, Fuzhou 350118, China, and also with the Fujian Provincial Key Laboratory of Network Computing and Intelligent Information Processing, Fuzhou 350118, China (e-mail: wengjiefzu@163.com; chenxing@fzu.edu.cn).

Yun Ma is with the Institute for Artificial Intelligence, Peking University, Beijing 100871, China (e-mail: mayun@pku.edu.cn).

Ching-Hsien Hsu is with the Department of Computer Science and Information Engineering, Asia University, Taichung 41354, Taiwan, and also with the Department of Medical Research, China Medical University Hospital, China Medical University, Taichung 404, Taiwan (e-mail: robertchh@gmail.com).

Digital Object Identifier 10.1109/TSC.2024.3517336

## I. Introduction

With the rapid development of 5G mobile communication technology and Internet of Things (IoT), many IoT applications, such as smart transportation, environment monitoring, e-sport, e-health, etc., have emerged recently [1]. Computation-intensive tasks generated by these applications in general require abundant computing resources to ensure the high service performance [2], [3]. However, due to the limited battery life and physical size constraint, IoT mobile devices (MDs) are unable to provide sufficient computing resources for efficiently task processing to meet the applications' requirements for low-latency and high-reliability services.

Mobile Cloud Computing (MCC) [4], [5] is considered as a promising solution for address above problems. It suggests moving computation-intensive tasks from MDs to the cloud, which can provide a large amount of computing resources to efficiently handle these tasks. However, considerable transmission delay will be incurred due to the geographically long distance between the cloud and MDs, so the QoS requirements of latency-sensitive applications cannot be always satisfied [6].

Mobile Edge Computing (MEC) [7], [8] is a novel computing paradigm to relive the tension between the insufficient computing resources of MDs and the high computing capacity requirements of computation-intensive tasks. It provides a better processing environment for the tasks of MDs by leveraging the edge resources. A MD can offload its tasks to the MEC server via wireless link (e.g., 5G or WiFi) between it and the MEC-enabled base station (BS) [9]. The MEC server offers much more computing capacity than the MD and are closer to the MD than the cloud, so it can obtain a high-quality and low-latency computing service for the computation-intensive tasks, resulting in more efficient task processing with lower transmission latency.

Due to the limited computing resources in a MEC-enabled BS, a large number of tasks offloaded to the BS from the MDs within its coverage cannot be processed in time during peak hours, which ultimately leads to system overload. The task processing performance of an overloaded MEC-enabled BS is severely degraded, and the user experience is further deteriorated. To mitigate this problem, most existing methods attempt to reduce the load on the BS by queuing, postponing, or rejecting the offloading requests from the covered MDs, resulting in impaired functionality of the corresponding applications.

To address this problem, one possible solution is to effectively balance the system load by migrating the high-load BS's tasks to its connected BSs to improve the system performance [10].

However, the decentralized locations of BSs and the limited resources of each MEC server make achieving load balancing among different BSs a difficult mission. Some research works focus on synergistic computation offloading among MEC servers through centralized decision-making methods [11], [12], [13], [14]. They usually employ heuristic-based searching strategies to find a suitable load balancing scheme. These strategies typically require a significant amount of searching time to reach a converged optimized solution, which is difficult to meet the real-time service requirements of most IoT applications. Besides, there are only a few works focusing on synergistic computation offloading through distributed decision-making methods [15], [16], [17], [18], which are less time-consuming to search a feasible solution. However, they typically make the offloading decisions according to all MEC servers' perfect information [19], which increases the offloading solution space. The larger solution space raises the likelihood to fall into poorer local optimal solutions, leading to a worse performance of the synergistic computation offloading in multi-edge environments.

The above works have a good inspiration for the synergistic computation offloading with imperfect information in multi-edge environments considering the partial offloading mode. However, they suffer from the following limitations.

- MEC servers from different BSs are typically selfish and rational, and each server has its specific goal. It is difficult to fully account for the individual benefits of each MEC server during computation offloading, which is detrimental to the optimization of overall system performance. Existing strategies mostly optimize the system performance from a global perspective, which are not suitable for the computation offloading among servers considering individual rationality.
- Existing works mostly make synergistic offloading decisions among servers through centralized methods, which require a lot of time to find the optimized solution and cannot be directly applied to the situations where the task load changes dynamically.
- Distributed decision-making methods with perfect information may fall into poorer local optimal solutions due to the large decision space, resulting in a worse performance of the obtained synergistic computation offloading scheme.

Motivated by the limitations of the existing literature, the main contributions of this article include:

- The synergistic computation offloading among MEC servers with equal status is modeled as a non-cooperative game with imperfect information, where servers make selfish and rational decisions according to the imperfect information about their adjacent servers. The game is then mathematically analyzed to prove that the Nash Equilibrium (NE) point can be obtained.
- A Synergistic Balancing Offloading Algorithm (SBOA) through distributed decision-making manner is proposed to obtain the optimal offloading strategy with a NE for all MEC servers in multi-edge environments. SBOA generates the optimal offloading vector for each server with convex optimization theory.
- Extensive experiments are conducted to evaluate the performance of SBOA. Simulation results show the SBOA has little execution time overhead and can quickly converge to the optimal load-balancing solution in multi-edge environment. In large-scale scenario, the results obtained by SBOA for the synergistic computation offloading are better than that of LC, IPA, DRL, and PSOGA by 15.2%, 3.4%, 4.2%, and 3.0%, respectively, in terms of system completion time.

The rest of this article is organized as follows. The related work is reviewed in Section II. Section III displays our problem formulation. Section IV discusses the construction of a non-cooperative game with imperfect information, and proves the existence of NE point for the game. Section V shows a detailed analysis for our proposed SBOA. Section VI conducts simulation evaluation and analysis. Finally, the work of this article is outlined in Section VII.

## II. Related Work

In this section, we briefly review the research on computation offloading between IoT devices and MEC servers, synergistic offloading among MEC servers, distributed offloading between servers with perfect information, and computation offloading between servers with imperfect information.

Many research efforts have been launched to computation offloading between IoT devices and MEC servers. IoT devices can effectively improve their performance by offloading tasks to MEC servers. Chen et al. [20] designed an efficient three-step scheme consisting of alternating optimization, sequential tuning, and semi-definite relaxation, which aimed to jointly optimize the offloading decisions of IoT devices' tasks as well as the resource allocation of servers' computation and communication in mobile cloud, to reduce the overall cost of computation, delay, and energy for IoT devices. Lu and Zhang [21] addressed a computation offloading problem for partitionable applications in dense networks. They jointly optimized the computation and radio resources, and aimed to minimize the task completion time. Extensive simulation results had showed the proposed mechanism performed better than the centralized optimal solution. Zhou et al. [22] investigated a UAV-aided computation offloading problem in MEC networks. They modeled the interaction among the edge service provider (ESP) and mobile users (MUs) through the Stackelberg game, and employed the backward induction approach to analyze it to maximize the utility of the ESP. The extensive evaluation results demonstrated the effectiveness of the proposed algorithm compared with other benchmark methods. Liu et al. [23] presented a flexible partial offloading strategy to adaptively meet each user's specific requirement regarding energy consumption and delay. They considered the practical variations in the user request patterns, and proposed an iterative algorithm to minimize the overheads subject to power constraints, task workload, and tolerable delay. Zhu et al. [24] proposed a deep reinforcement learning-based edge computing offloading algorithm for software-defined IoT. This algorithm could effectively reduce energy consumption and task completion time compared with other classical methods. Chen et al. [25] designed an energy efficient task offloading algorithm for digital twin-empowered MEC via deep reinforcement learning, to improve energy efficiency and balance the

workload. The above methods reduce the server load through queuing, postponement, or rejection schemes when receiving too many tasks, which will sacrifice the quality of experience for IoT devices.

To address above issues, some studies have focused on synergistic offloading among multiple MEC servers to achieve load balancing. These solutions can reduce the response latency of high-load MEC servers, and improve the user experience of IoT devices, through fully utilizing the arithmetic power of servers. Diamanti et al. [26] investigated the application of the RSMA technique to facilitate the users' concurrent offloading to multiple servers in a multi-server MEC system. Yi et al. [27] studied a long-term workload management problem for multi-server edge computing with server collaboration, and considered both competitions and collaborations among strategic edge servers in sharing their computing capacities. They presented a novel cooperative queueing game approach, which outperformed other counterparts. Yang et al. [28] presented the differential evolution (DE) based multi-UAV deployment scheme to balance the load for UAVs. The experimental results revealed that the superiority and feasibility of the proposed scheme. Wan et al. [29] designed an energy-aware load balancing and scheduling scheme based on fog computing in smart factory. Simulation results showed that the method performed optimal load balancing and scheduling for the mixing work robots. Lin et al. [30] proposed a self-adaptive discrete particle swarm optimization algorithm with genetic algorithm operators to optimize the data transmission time among servers. The above studies mainly employ centralized approaches to obtain the load balancing solution among servers, which may cost much time and is not suitable for scenarios with fluctuating loads.

Distributed algorithms [31] can alleviate the above problems by giving equal decision-making power to MEC servers, which effectively reduces the time taken to obtain solutions. However, there is non-cooperative competition among servers, which makes the offloading problem more complicated. Liu et al. [15] formulated the multi-server load balancing problem into a non-cooperative game, and designed an iterative proximal algorithm (IPA) to reduce the response time of all servers. Simulation results revealed that the IPA converged quickly to a NE point and performed better than other benchmark algorithms. Fan et al. [16] presented a game-based multitype task offloading strategy among MEC-enabled BSs. They formulated the task offloading with different types (indicated by delay tolerance, data size, and computation amount) into a non-cooperative game, and proposed a distributed iterative algorithm to balance the delays of all tasks from each MEC-BS. Xu et al. [18] introduced a distributed algorithm with polynomial time to address the distributed assignment with load balancing for deep neural networks (DNN) inference at the edge. Extensive simulations showed the solution could improve upon the state-of-the-art in terms of load balance, inference time, and convergence. In these studies, MEC servers need to know perfect information about other all servers before they make the computation offloading decisions. However, these distributed algorithms with perfect information may fall into poorer local optimal solutions due to the large decision space.

Compared to the perfect information-based approaches, the core challenge encountered by imperfect information-based approaches is how to optimize the algorithms to ensure the system performance under information-constrained conditions, in order to minimize the performance degradation caused by imperfect information. Hu et al. [32] addressed the heterogeneous task offloading with imperfect information in a distributed edge computing system. They formulated this problem as a multi-player minority game (MG), and proposed an MG based scheme to reduce the task processing time. They proved that the proposed algorithm could converge to a near-optimal point, and its superior performance. Wang et al. [33] designed an incomplete information based two-tier game (IITG) model to realize collaborative computing at the edge of emergency communication networks. They also proposed a near-optimal IITG algorithm (N-IITG) to seek the unique Bayesian Nash equilibrium, which incentivized idle computing devices to share computation resources. Extensive simulations showed N-IITG has better performance compared with other classical methods. The above methods mainly adopts the master-slave architecture for the deployment of computation offloading policies (i.e., the master node is responsible for the formulation and allocation of computation offloading policies, and the slave nodes carry out computation offloading according to the policies formulated by the master node). This system architecture has limitations in terms of system reliability and fault tolerance. Once the master node fails or malfunctions, the system loses the ability to formulate computation offloading policies, resulting in the inability of the slave nodes to perform effective load balancing.

Table I has compared some related work with our work. We categorize the computation offloading strategies in seven dimensions: Environment, Surrounding perception, Interaction, Offloading mode, Execution manner, Object and Method. The environment includes device-edge, multi-edge, and device-edge-cloud scenarios. The surrounding perception are divided into perfect information and imperfect information. The interactions among servers are divided into cooperative and non-cooperative games. The offloading modes contain full offloading and partial offloading. The execution manners are mainly divided into centralized manner and distributed manner, where the decision maker is device or edge. The most common optimization objectives are cost and time. Finally, we classify the offloading strategies into heuristics, mathematical programming, machine learning and game. Despite the aforementioned prominent developments in recent years, a considerable open issue remains on computation offloading in multi-edge environments: Synergistic computation offloading through distributed decision-making manner among MEC servers with imperfect information. Inspired by above observations, this article proposes a game-based computation offloading scheme in imperfect-information multi-edge environments.

## III. Problem Formulation

The main symbols involved in this paper are shown in Table II.

In this section, we formulate the problem of synergistic computation offloading in multi-edge environments. As shown in

TABLE I
COMPARISON OF OUR WORK WITH RELATED WORK

| Reference | Environment | | | Surrounding perception | | Iteraction | | Offloading mode | | Execution manner | | | Object | | | Method | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Device - Edge | Multi-edge | Device - Edge-Cloud | Perfect Information | Imperfect Information | Cooperative game | Non-cooperative game | Full Offloading | Partial Offloading | Centralized | Distributed (Decision maker) | | Cost | Time | Heurisities | Mathematical programming | Machine Learning | Game |
| | | | | | | | | | | | Device | Edge | | | | | | |
| [20] | - | - | + | + | - | - | - | - | + | + | - | - | + | + | + | + | - | - |
| [21] | + | - | - | - | + | - | + | + | - | - | + | - | - | + | - | - | - | + |
| [22] | + | - | - | - | + | - | + | - | + | - | + | - | + | + | - | - | - | + |
| [23] | + | - | - | + | - | - | - | - | + | + | - | - | + | + | - | + | - | - |
| [24] | - | + | - | - | + | - | - | + | - | + | - | - | + | + | - | - | + | - |
| [25] | + | - | - | - | + | - | - | + | - | - | + | - | + | + | - | - | + | - |
| [26] | - | + | - | + | - | - | - | + | - | + | - | - | - | + | - | + | - | - |
| [27] | - | + | - | + | - | + | - | - | + | - | - | + | + | + | - | + | - | + |
| [28] | + | - | - | + | - | - | - | + | - | - | - | + | - | + | + | - | + | - |
| [29] | + | - | - | + | - | - | - | - | - | + | - | - | - | + | + | - | - | - |
| [30] | - | - | + | + | - | - | - | - | - | + | - | - | - | + | + | - | - | - |
| [31] | + | - | - | - | + | - | + | + | - | - | + | - | - | + | - | - | + | + |
| [15] | - | + | - | + | - | - | + | + | - | - | - | + | - | + | - | - | - | + |
| [16] | - | + | - | + | - | - | + | + | - | - | - | + | - | + | - | - | - | + |
| [18] | - | + | - | - | + | - | - | + | - | - | + | + | - | + | - | + | - | - |
| [32] | - | - | + | - | + | - | + | + | - | + | + | - | - | + | + | - | - | + |
| [33] | + | - | - | - | + | - | + | + | - | - | + | - | - | + | - | - | - | + |
| Our work | - | + | - | - | + | - | + | - | + | - | - | + | - | + | - | + | - | + |

TABLE II
SYMBOLS AND DEFINITIONS

| Symbol | Definition |
|---|---|
| $n$ | Number of MEC servers |
| $\boldsymbol{E}$ | Set of MES servers |
| $e_i$ | the $i$th MEC server |
| $f_i$ | $e_i$'s service rate |
| $\boldsymbol{D}$ | Transmission time per unit of task volume between MEC servers |
| $d_{i,j}$ | The time it takes for $e_i$ to offload the unit task volume to $e_j$ |
| $\boldsymbol{A}_i$ | $e_i$'s adjacent servers as well as its own |
| $\lambda_i$ | Average task arrival rate of $e_i$ |
| $s_i$ | Aggregation task arrival rate of $e_i$ |
| $\boldsymbol{X}$ | Offloading strategy for all MEC servers |
| $\boldsymbol{X}_i$ | Offloading vector of $e_i$ |
| $\boldsymbol{X}_{-i}$ | $e_i$'s adjacent servers' offloading decisions |
| $x_{i,j}$ | Task volume offloaded from $e_i$ to $e_j$ per time unit |
| $t_i^{loc}$ | Average completion time of $e_i$ before balancing offloading |
| $t_i^{off}$ | Average completion time of $e_i$ after balancing offloading |
| $Q_i$ | Disutility function of $e_i$ |
| $\boldsymbol{O}_i$ | $e_i$'s offloading server vector |

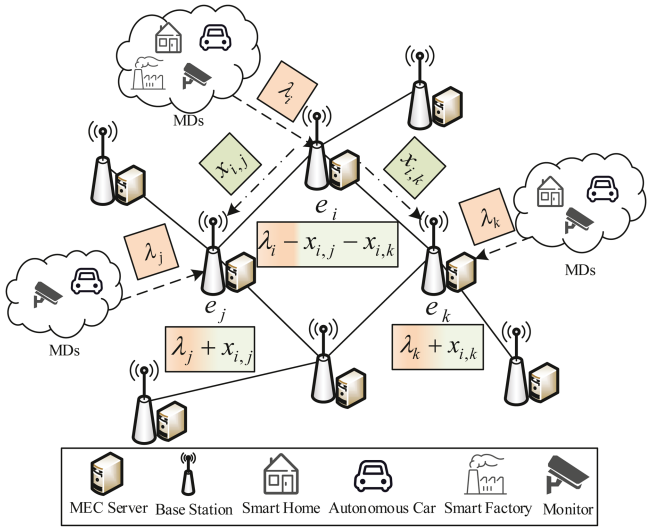



Fig. 1. Framework of synergistic computation offloading in multi-edge environments.

Fig. 1, we consider a scenario consisting of multiple BSs, each of which is equipped with a MEC server. These MEC servers with equal status have different computational capabilities, and provide computation offloading services for the IoT MDs covered by the corresponding BS. IoT MDs can offload their tasks to BSs for auxiliary computing, thus improving the task processing performance.

There are $n$ MEC servers in the multi-edge environments, $\boldsymbol{E} = \{e_1, e_2, \ldots, e_n\}$, where $e_i$ denotes the $i$th server. $f_i$ is $e_i$'s service rate, which represents the task volume that can be processed by server $e_i$ per time unit. Each MEC server is interconnected with its adjacent servers via wired fiber connections, which provide bi-directional data transmission with the same bandwidth. Too many information interactions among servers can lead to a large number of offloading options for each server, increasing the offloading solution space and the risk of falling into poor local optimal solutions for the final offloading strategy [34]. Therefore, the information sharing among MEC

servers is restricted in our model. Specially, each MEC server only has access to the information of its adjacent servers (i.e., imperfect information), instead of the information of all servers (i.e., perfect information) in the multi-edge environments.

The transmission time per unit of task volume between MEC servers [35] is defined as

$$\boldsymbol{D} = \begin{bmatrix} d_{1,1} & \cdots & d_{1,n} \\ \vdots & \ddots & \vdots \\ d_{n,1} & \cdots & d_{n,n} \end{bmatrix}, \tag{1}$$

where $d_{i,j}$ denotes the time it takes for server $e_i$ to offload the unit task volume to server $e_j$. Note that $d_{i,j} = d_{j,i}$, and $d_{i,j} = 0$ when $i = j$. If server $e_i$ and server $e_j$ are not adjacent servers, $d_{i,j} = +\infty$. In addition, the feasible server vector of server $e_i$, $\boldsymbol{A}_i = \{e_j | d_{i,j} \neq +\infty\}$, denotes all servers that can perform the tasks on server $e_i$ (i.e., $e_i$'s adjacent servers as well as its own).

The average task arrival rate of server $e_i$ is defined as $\lambda_i$, which represents the task volume offloaded from IoT MDs to server $e_i$ per time unit. Tasks can be performed locally or remotely on adjacent servers through computation offloading. We assume that each MEC server can arbitrarily partition the tasks from IoT MDs and schedule them to other adjacent servers [36]. $s_i$ is the aggregation task arrival rate of server $e_i$, which contains the arrival tasks executed by $e_i$ itself as well as the tasks offloaded to it by other adjacent servers per time unit. The offloading strategy for all MEC servers is defined as

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{X}_1 \\ \vdots \\ \boldsymbol{X}_n \end{bmatrix} = \begin{bmatrix} x_{1,1} & \cdots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,n} \end{bmatrix}, \tag{2}$$

where $\boldsymbol{X_i}$ is the offloading vector of server $e_i$, which represent $e_i$'s balancing offloading plan. Note that a MEC server can only offload its own arrival tasks, and cannot schedule tasks offloaded from other servers. $x_{i,j}$ denotes the task volume offloaded from server $e_i$ to server $e_j$ per time unit. When $i = j$, $x_{i,j}$ represents $e_i$'s arrival task volume performed by server $e_i$ itself per time unit. Specially, $x_{i,j} = 0$ if $e_j \notin \boldsymbol{A}_i$.

In our work, the process queue on MEC servers regards as an $M/M/1$ queue model [37]. Before synergistic computation offloading, all arriving tasks on servers are executed locally and the load on each server is equal to its amount of arriving tasks (i.e., $s_i = \lambda_i$). Therefore, the average completion time $t_i^{loc}$ of all arriving tasks on $e_i$ before balancing offloading is represented as

$$t_i^{loc} = \frac{1}{\lambda_i}\left[\frac{s_i}{f_i \cdot (f_i - s_i)} + \frac{s_i}{f_i}\right]. \tag{3}$$

After synergistic computation offloading, the tasks on server $e_i$ include the arrival tasks executed by $e_i$ itself as well as the tasks offloaded to it by other adjacent servers (i.e., $s_i = x_{i,i} + \sum_{j=1, j\neq i}^{n} x_{j,i}$). In addition, the computation offloading between MEC servers incurs additional communication overheads. Therefore, the average completion time $t_{i,j}^{off}$ of tasks offloaded from server $e_i$ to server $e_j$ can be expressed as

$$t_{i,j}^{off} = \frac{1}{\lambda_i}\left[\frac{s_j}{f_j \cdot (f_j - s_j)} + \frac{x_{i,j}}{f_j} + x_{i,j} \cdot d_{i,j}\right], \tag{4}$$

$$t_i^{off} = \sum_{e_j \in \boldsymbol{O}_i} t_{i,j}^{off}, \tag{5}$$

where $\boldsymbol{O}_i \subseteq \boldsymbol{A}_i$ is the $e_i$'s offloading server vector, and $e_j \in \boldsymbol{O}_i$ means that $e_i$ must has tasks to offload to $e_j$. $t_i^{off}$ is the average completion time of $e_i$ after balancing offloading.

Our goal is to minimize each server's average completion time after synergistic computation offloading in multi-edge environments. Therefore, the disutility function of server $e_i$ in our imperfect-information model is defined as

$$Q_i = \frac{1}{\lambda_i}\left\{\sum_{e_j \in \boldsymbol{O}_i}\left[\frac{s_j}{f_j \cdot (f_j - s_j)} + \frac{x_{i,j}}{f_j}\right] + \sum_{e_j \in \boldsymbol{O}_i} x_{i,j} \cdot d_{i,j}\right\}. \tag{6}$$

The goal of each server $e_i$ in the multi-edge environments is to dynamically offload the tasks, according to the current network state and the adjacent servers' offloading decisions $\boldsymbol{X}_{-i}$, to find the optimal offloading decision that minimizes the disutility function of the server $e_i$ itself. Therefore, the corresponding optimization problem of server $e_i$ can be defined as (7).

$$\begin{aligned} &\text{Minimize } Q_i\left(\boldsymbol{X}_i, \boldsymbol{X}_{-i}\right) \\ &\text{s.t. } C1: x_{i,j} \geq 0 \\ &\qquad C2: \sum_{j=1}^{n} x_{i,j} = \lambda_i \\ &\qquad C3: \sum_{i=1}^{n} x_{i,j} < f_j, \end{aligned} \tag{7}$$

where the third constraint $C3$ ensures that the offloaded task volume to server $e_j$ per unit time should be less than its service rate.

Each server should have incentives for computation offloading [38], and the designed reward scheme $R_i$ for server $e_i$ is defined as

$$R_i = -Q_i, \tag{8}$$

where the process of reducing disutility (i.e., reducing completion time) for each server essentially constitutes rationality incentives for servers in our work.

## IV. Game Formulation and Proof

In this section, we first construct a non-cooperative game with imperfect information to achieve the efficient balancing offloading strategies among MEC servers, and then prove the existence of NE point for the game.

### A. Game Formulation

It is usually categorized into cooperative and non-cooperative games according to the relationship between the players in the game theory [39]. The difference between cooperative and non-cooperative games is whether the players form a binding coalition or not, which will affect the players' decisions. Specifically, the cooperative game takes full account of the fair position

among the players, whereas players in the non-cooperative game are selfish and they prioritize maximizing their own benefits.

In addition, it is also categorized into perfect-information and imperfect-information games depending on the degree of information sharing among the players [40]. In the perfect-information game, each player has perfect information about other players' characteristics, benefit functions, and strategy sets, whereas each player does not have such perfect information in the imperfect-information game.

In multi-edge environments, each MEC server is regarded as a player in a game, whose goal is to make an optimal offloading strategy to reduce the average completion time of its arriving tasks. Server $e_i$'s average completion time is not only dependent on its own offloading strategy, but also influenced by other servers' offloading strategies. This is because server $e_i$ can execute its tasks locally, or offload them to its adjacent servers, and also has to execute the tasks offloaded from its adjacent servers.

In the game, each MEC server is selfish and tries to optimize their goals (i.e., minimizing the disutility function) according to the local information of its adjacent servers. Therefore, we define this problem as a non-cooperative game with imperfect-information, where the players (i.e., servers) have fair positions. The game consists of a set of players, a set of strategies, and a set of disutility functions, which is expressed as (9).

$$\boldsymbol{G} =< \boldsymbol{E}, \{\boldsymbol{X}_i\}_{e_i \in \boldsymbol{E}}, \{Q_i\}_{e_i \in \boldsymbol{E}} >, \tag{9}$$

where $\boldsymbol{E}$ is the set of $n$ MEC servers (i.e., players), $\boldsymbol{X}_i$ is the server $e_i$'s offloading vector (i.e., strategy set), and $Q_i$ is the server $e_i$'s disutility function.

In game theory, Nash Equilibrium (NE) is employed as an important condition to measure the stability of a system. In our work, the definition of the NE point is described below.

*Definition 1 (NE point):* The NE point of the game $\boldsymbol{G}$ is an achievable offloading strategy $\boldsymbol{X}^* = \{\boldsymbol{X}_1^*, \ldots, \boldsymbol{X}_n^*\}$ for all MEC servers, where any server's strategy $\{\boldsymbol{X}_i^*\}_{e_i \in \boldsymbol{E}} \subseteq \boldsymbol{X}^*$ satisfies the following conditions.

$$Q_i\left(\boldsymbol{X}_i^*, \boldsymbol{X}_{-i}^*\right) \leq Q_i\left(\boldsymbol{X}_i', \boldsymbol{X}_{-i}^*\right), \tag{10}$$

where $\{\boldsymbol{X}_i'\}_{e_i \in \boldsymbol{E}}$ is an arbitrary strategy of the server $e_i$, and $\boldsymbol{X}_{-i}^* = \{\boldsymbol{X}_1^*, \ldots, \boldsymbol{X}_{i-1}^*, \boldsymbol{X}_{i+1}^*, \ldots, \boldsymbol{X}_n^*\}$.

According to Definition 1, no player can reduce its disutility by unilaterally changing its strategy when the system is at NE point, so no player has an incentive to deviate from the NE point.

### B. Existence of NE Point

The game is transformed into variational inequalities [39], and then the existence of NE point for the synergistic computation offloading in multi-edge environments will be proven.

*Theorem 1:* For each MEC server $e_i$, its offloading vector $\boldsymbol{X}_i$ (i.e., strategy set) is closed and convex, and its disutility function $Q_i$ is continuously differentiable.

*Proof:* In our system, each server $e_i$'s offloading vector $\boldsymbol{X}_i$ satisfies constraints $C1 - C3$ in (7), where $\{\boldsymbol{X}_i | \forall x_{i,j} \in [0, \lambda_i]\}$. According to the definition of a closed convex set [23], it is obvious that the offloading vector $\boldsymbol{X}_i$ (i.e., strategy set) is closed and convex.

For the latter half, we should prove the derivative function's continuity of the disutility function $Q_i$. Since $Q_i$ is a multivariate nonlinear function, the gradient $\boldsymbol{q}$ is defined as (11), which represents the combination of partial derivatives in each direction for $Q_i$.

$$\boldsymbol{q}\left(\boldsymbol{X}_i, \boldsymbol{X}_{-i}\right) \triangleq \left(q_i\left(\boldsymbol{X}_i, \boldsymbol{X}_{-i}\right)\right)_{i=1}^n = \left(\nabla_{\boldsymbol{X}_i} Q_i\left(\boldsymbol{X}_i, \boldsymbol{X}_{-i}\right)\right)_{i=1}^n, \tag{11}$$

where the expression of $q_i(\boldsymbol{X}_i, \boldsymbol{X}_{-i})$ is described as (12) based on the definition of the gradient [41]. In addition, $\boldsymbol{q}(\boldsymbol{X}_i) = \boldsymbol{q}(\boldsymbol{X}_i, \boldsymbol{X}_{-i})$ and $q_i(\boldsymbol{X}_i) = q_i(\boldsymbol{X}_i, \boldsymbol{X}_{-i})$.

$$\begin{aligned} q_i\left(\boldsymbol{X}_i, \boldsymbol{X}_{-i}\right) &= \nabla_{\boldsymbol{X}_i} Q_i\left(\boldsymbol{X}_i, \boldsymbol{X}_{-i}\right) \\ &= \left(\frac{\partial Q_i}{\partial x_{i,1}}, \ldots, \frac{\partial Q_i}{\partial x_{i,n}}\right). \end{aligned} \tag{12}$$

Then, server $e_i$'s disutility function $Q_i$ can be rewritten as (13).

$$\begin{aligned} Q_i &= \frac{1}{\lambda_i}\left\{\sum_{e_j \in \boldsymbol{O}_i}\left[\frac{s_j}{f_j \cdot (f_j - s_j)} + \frac{x_{i,j}}{f_j}\right] + \sum_{e_j \in \boldsymbol{O}_i} x_{i,j} \cdot d_{i,j}\right\} \\ &= \frac{1}{\lambda_i}\left\{\sum_{e_j \in \boldsymbol{O}_i}\left[\frac{1}{f_j - s_j} - \frac{1}{f_j} + \frac{x_{i,j}}{f_j}\right] + \sum_{e_j \in \boldsymbol{O}_i} x_{i,j} \cdot d_{i,j}\right\}. \end{aligned} \tag{13}$$

The partial derivative of $Q_i$ with respect to $x_{i,j}$ is

$$\frac{\partial Q_i}{\partial x_{i,j}} = \begin{cases} \frac{1}{\lambda_i}\left(\frac{1}{(f_j - s_j)^2} + \frac{1}{f_j} + d_{i,j}\right), & \text{if } i \neq j \\ \frac{1}{\lambda_i}\left(\frac{1}{(f_j - s_j)^2} + \frac{1}{f_j}\right), & \text{else} \end{cases}. \tag{14}$$

Since the partial derivatives of $Q_i$ exist and are continuous, it is obtained that the disutility function $Q_i$ is continuously differentiable.

*Theorem 2:* The disutility function of MEC server $e_i$ is convex when other servers no longer change their offloading strategies.

*Proof:* According to the definition of a convex function [42], the disutility function $Q_i$ is convex if its Hessian matrix $\boldsymbol{H}(Q_i)$ is positive definite.

$$\boldsymbol{H}(Q_i) = \begin{bmatrix} \frac{\partial^2 Q_i(\boldsymbol{X}_i, \boldsymbol{X}_{-i})}{\partial^2 x_{i,1}} & \cdots & \frac{\partial^2 Q_i(\boldsymbol{X}_i, \boldsymbol{X}_{-i})}{\partial x_{i,1} \partial x_{i,n}} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 Q_i(\boldsymbol{X}_i, \boldsymbol{X}_{-i})}{\partial x_{i,n} \partial x_{i,1}} & \cdots & \frac{\partial^2 Q_i(\boldsymbol{X}_i, \boldsymbol{X}_{-i})}{\partial^2 x_{i,n}} \end{bmatrix}, \tag{15}$$

$$\frac{\partial^2 \boldsymbol{Q}_i\left(\boldsymbol{X}_i, \boldsymbol{X}_{-i}\right)}{\partial x_{i,j} \partial x_{i,k}} = \begin{cases} \frac{2}{(f_j - s_j)^3}, & \text{if } i = j = k \\ 0, & \text{else} \end{cases}. \tag{16}$$

When other servers no longer change their offloading strategies, the Hessian matrix $\boldsymbol{H}(Q_i)$ of server $e_i$'s disutility function $Q_i$ is a diagonal matrix with diagonal elements of $\frac{2}{(f_j - s_j)^3}$. In our system, all MEC servers should satisfy the third constraint $C3$ in (7), which means that $f_j - s_j > 0$. Accordingly, $\frac{2}{(f_j - s_j)^3} > 0$

and the Hessian matrix $\boldsymbol{H}(Q_i)$ is positive definite. Therefore, the disutility function of MEC server $e_i$ is convex when other servers no longer change their offloading strategies, according to the definition of a convex function.

*Theorem 3:* If $\boldsymbol{X}_i$ is a nonempty closed convex set and $Q_i$ is a continuously differentiable convex function, then solving the game $\boldsymbol{G} =< \boldsymbol{E}, \{\boldsymbol{X}_i\}_{e_i \in \boldsymbol{E}}, \{Q_i\}_{e_i \in \boldsymbol{E}} >$ is equivalent to finding a solution to the variational inequality $\mathrm{VI}(\boldsymbol{X}, \boldsymbol{q})$.

*Proof:* Obviously, $\boldsymbol{X}_i$ is a nonempty set. According to the proofs for Theorems 1 and 2, $\boldsymbol{X}_i$ is a nonempty closed convex set and $Q_i$ is a continuously differentiable convex function for our game. Scutari et al. [43] have discussed the intrinsic relationship between game theory and variational inequalities. According to the analysis results, if the first part in Theorem 3 holds, solving the game $\boldsymbol{G} =< \boldsymbol{E}, \{\boldsymbol{X}_i\}_{e_i \in \boldsymbol{E}}, \{Q_i\}_{e_i \in \boldsymbol{E}} >$ is equivalent to finding a solution to the variational inequality $\mathrm{VI}(\boldsymbol{X}, \boldsymbol{q})$. Therefore, Theorem 3 holds for our game.

*Theorem 4:* The gradient $\boldsymbol{q}$ in $\mathrm{VI}(\boldsymbol{X}, \boldsymbol{q})$ is strictly monotone.

*Proof:* If the conditions in (17) are satisfied, gradient $\boldsymbol{q}$ is strictly monotone.

$$\begin{aligned}&(\boldsymbol{X}-\boldsymbol{X}^*)^T \cdot (\boldsymbol{q}(\boldsymbol{X}) - \boldsymbol{q}(\boldsymbol{X}^*)) \\ &\quad = (\boldsymbol{X}_1 - \boldsymbol{X}_1^*, \ldots, \boldsymbol{X}_n - \boldsymbol{X}_n^*)^T \cdot \\ &(q_1(\boldsymbol{X}_1) - q_1(\boldsymbol{X}_1^*), \ldots, q_n(\boldsymbol{X}_n) - q_n(\boldsymbol{X}_n^*)) \geq 0, \end{aligned} \tag{17}$$

which is equivalent to (18).

$$\sum_{i=1}^{n} (\boldsymbol{X}_i - \boldsymbol{X}_i{}^*)^T \cdot (q_i(\boldsymbol{X}_i) - q_i(\boldsymbol{X}_i{}^*)) \geq 0. \tag{18}$$

For $\forall i \in n$, if

$$(\boldsymbol{X}_i - \boldsymbol{X}_i{}^*)^T \cdot (q_i(\boldsymbol{X}_i) - q_i(\boldsymbol{X}_i{}^*)) \geq 0, \tag{19}$$

then (18) holds.

If the Jacobian matrix $\boldsymbol{J}_{q_i}(\boldsymbol{X}_i)$ for $q_i$ is positive definite, then (19) holds. Eqution (20) is obtained according to the definition of the Jacobian matrix [44].

$$\boldsymbol{J}_{q_i}(\boldsymbol{X}_i) = (\nabla^2_{\boldsymbol{X}_i \boldsymbol{X}_i} \cdot Q_i(\boldsymbol{X}_i)) = \boldsymbol{H}(Q_i). \tag{20}$$

Theorem 2 has proved that the Hessian matrix $\boldsymbol{H}(Q_i)$ is positive definite, so the Jacobian matrix $\boldsymbol{J}_{q_i}(\boldsymbol{X}_i)$ for $q_i$ is also positive definite. Therefore, the gradient $\boldsymbol{q}$ in $\mathrm{VI}(\boldsymbol{X}, \boldsymbol{q})$ is strictly monotone.

According to the analysis results [39], if the gradient $\boldsymbol{q}$ is strictly monotone, there is at least one NE point for the game. In summary, we prove the existence of NE point for our game.

## V. Our Proposed SBOA

In the non-cooperative game with imperfect information for synergistic computation offloading in multi-edge environments, the players are self-interested and their status during computation offloading is equal. They make offloading decisions independently according to the imperfect information from their adjacent servers. They only focus on optimizing their own goals and do not take into account the impact of their own strategy changes on other players. Due to the existence of NE point for our game, we minimize the disutility function of each player to obtain its optimal offloading vector iteratively.

**Algorithm 1:** Synergistic Balancing Offloading Algorithm (SBOA).

**Input:** $f_i, \lambda_i, \boldsymbol{D}, \boldsymbol{A}_i$.
**Output:** $\boldsymbol{X}^*$.
1: **Initialization**: $\boldsymbol{X}^{(0)} \leftarrow diag(\{\lambda_i\}_{i=1}^n), k \leftarrow 0$;
2: Each server broadcasts its load information to the adjacent servers;
3: **while true do**
4: $k \leftarrow k + 1$;
5: **foreach** $e_i \in \boldsymbol{E}$ **do**
6: Obtain $\boldsymbol{O}_i^*$ through Algorithm 2;
7: Based on variables $\boldsymbol{O}_i^*, f_i, \lambda_i, \boldsymbol{D}$ and $\boldsymbol{X}_{-i}$, using CVX to obtain $\boldsymbol{X}_i^{(k)}$, where $\boldsymbol{X}_i^{(k)} = \mathrm{argmin}\{Q_i(\boldsymbol{X}_i, \boldsymbol{X}_{-i})\}$;
8: **end**
9: **if** $\boldsymbol{X}^{(k)} = \boldsymbol{X}^{(k-1)}$ **then**
10: **break**;
11: **end**
12: **end**
13: $\boldsymbol{X}^* \leftarrow \boldsymbol{X}^{(k)}$;
14: **return** $\boldsymbol{X}^*$;

A Synergistic Balancing Offloading Algorithm (SBOA) is proposed to obtain the optimal offloading strategy with a NE for all MEC servers. The algorithm is executed in a distributed manner on all MEC servers in turn, which tries to generate the optimal offloading vector for each server. It adopts the distributed architecture for the deployment of computation offloading policies, which could avoid the potential threat of single-point-of-failure to the stability and performance of the entire system. The pseudocode of SBOA is described as Algorithm 1. In the beginning, each server's offloading vector is equal to its amount of arriving tasks (i.e., computing locally), and the number of current iteration is initialized to 0 (line 1). Then, each MEC server broadcasts its load information to their adjacent servers (line 2). In each iteration, each server $e_i$ obtains its optimal offloading server vector $\boldsymbol{O}_i^*$ through Algorithm 2, and utilizes the CVX toolkit [45] to obtain its optimal offloading vector $\boldsymbol{X}_i^{(k)}$ in current iteration, where $\boldsymbol{X}_i^{(k)} = \mathrm{argmin}\{Q_i(\boldsymbol{X}_i, \boldsymbol{X}_{-i})\}$ (lines 4-8). This process uses a convex optimization method, which belongs to the mathematical optimization [46]. After all MEC servers obtain their optimal offloading vectors, a new iteration begins until all servers' offloading vectors are fixed. If all servers' offloading vectors in current iteration are consistent with those of the previous iteration, the algorithm reaches a NE point and exits the iteration (lines 9-11). Finally, $\boldsymbol{X}^{(k)}$ is assigned to $\boldsymbol{X}^*$ as the offloading strategy with a NE for all MEC servers (line 13).

A MEC server may not offload tasks to every adjacent server with an available offloading policy. It should have a suitable offloading server vector $\boldsymbol{O}_i$ to better optimize its disutility function based on its imperfect information. An Adaptive Selection Algorithm (ASA) is proposed to obtain server $e_i$'s optimal offloading server vector $\boldsymbol{O}_i^*$. The pseudocode of ASA

**Algorithm 2:** Adaptive Selection Algorithm (ASA).

**Input:** $f_i, \lambda_i, \boldsymbol{D}, \{\boldsymbol{X}_{-i}\}_{e_{-i}\in\boldsymbol{A}_i}, \boldsymbol{A}_i$
**Output:** $\boldsymbol{O}_i^*$
1: **Initialization**: $m \leftarrow |\boldsymbol{A}_i|, \boldsymbol{W} \leftarrow \emptyset$;
2: Generate $\boldsymbol{B} = \{b_1, b_2, \ldots, b_{2^m-1}\}$ according to $\boldsymbol{A}_i$, where $b_k = [0, \ldots, 1]_m$;
3: **foreach** $b_k \in \boldsymbol{B}$ **do**
4: $\quad\bar{\boldsymbol{O}}_i^k \leftarrow \emptyset$;
5: $\quad$**foreach** $b_k[r]$ **do**
6: $\quad\quad$**if** $b_k[r] = 1$ **then**
7: $\quad\quad\quad\bar{\boldsymbol{O}}_i^k \cup \{\boldsymbol{A}_{ir}\}$;
8: $\quad\quad$**end**
9: $\quad$**end**
10: $\quad$Based on variables $f_i, \lambda_i, \boldsymbol{D}, \{\boldsymbol{X}_{-i}\}_{e_{-i}\in\boldsymbol{A}_i}$ and $\bar{\boldsymbol{O}}_i^k$, using CVX to obtain $Q_i^k$, where $Q_i^k = \min\{Q_i(\boldsymbol{X}_i, \boldsymbol{X}_{-i})\}$;
11: $\quad\boldsymbol{W} \cup \{(\bar{\boldsymbol{O}}_i^k, Q_i^k)\}$;
12: **end**
13: $\boldsymbol{O}_i^* \leftarrow \bar{\boldsymbol{O}}_i^{\mathrm{argmin}_k\{Q_i^k|(\bar{\boldsymbol{O}}_i^k, Q_i^k)\in\boldsymbol{W}}$;
14: **return** $\boldsymbol{O}_i^*$;

is shown in Algorithm 2. In the beginning, the value of $m$ is initialized to the number of $e_i$'s adjacent servers (i.e., $|\boldsymbol{A}_i|$), and the pending set $\boldsymbol{W}$ is set to be empty (line 1). Then, the set $\boldsymbol{B} = \{b_1, b_2, \ldots, b_{2^m-1}\}$ is constructed, and it consists of $2^m - 1$ different 0/1 combinations $b_k = [0, \ldots, 1]_m$ of length $m$ (i.e., full-coverage combinations without $[0, \ldots, 0, 0]_m$) (line 2). Each bit of $b_k$ corresponds to whether the server at the same location of $\boldsymbol{A}_i$ is selected or not, where 1 means the server is selected and 0 means the opposite. Next, It iterates through the elements in $\boldsymbol{B}$ one by one. For $b_k$, each bit of which is checked. If the value is 1 at a position of $b_k$, the server corresponding to that position is added to $\bar{\boldsymbol{O}}_i^k$ (lines 4-9). The CVX toolkit [45] is utilized to obtain the minimum of the disutility function $Q_i^k$, where $Q_i^k = \min\{Q_i(\boldsymbol{X}_i, \boldsymbol{X}_{-i})\}$ (line 10). This process uses a convex optimization method, which belongs to the mathematical optimization [46]. The couple $(\bar{\boldsymbol{O}}_i^k, Q_i^k)$ in current iteration $k$ is added to $\boldsymbol{W}$ (line 11). After scanning all combinations of $\boldsymbol{B}$, it selects the $\bar{\boldsymbol{O}}_i^k$ corresponding to the minimum $Q_i^k$ in the couples of $\boldsymbol{W}$, and assigns it to $\boldsymbol{O}_i^*$. This process utilizes a greedy strategy to obtain the minimum $Q_i^k$, which belongs to the heuristics [47].

The time complexity of the above algorithms is discussed as follows. ASA (i.e., Algorithm 2) is part of SBOA (i.e., Algorithm 1). In Algorithm 2, the time complexity of the initialization process (line 1) is $O(1)$, and the generation of the set $\boldsymbol{B}$ with $2^m - 1$ different combinations (line 2) is $O(2^m \cdot m)$. In each **for** process (lines 3-12), the time complexity of constructing $\bar{\boldsymbol{O}}_i^k$ (lines 4-9) is $O(m)$. Supposing the time complexity of solving the problem of obtaining the minimum of the disutility function $Q_i^k$ with CVX toolkit (line 10) is $O(\sigma)$. Accordingly, the time complexity of the **for** process is $O(2^m \cdot (m + \sigma))$. Finally, the selection process of the $\bar{\boldsymbol{O}}_i^k$ corresponding to the minimum $Q_i^k$ in the couples of $\boldsymbol{W}$ (line 13) is $O(2^m - 1)$. Therefore, the time complexity of ASA is $O(1 + 2^m \cdot m + 2^m \cdot (m + \sigma) + 2^m - 1)$, which can be simplified to $O(2^m \cdot (m + \sigma))$.

In Algorithm 1, the time complexity of the initialization process for $\boldsymbol{X}$ and $k$ (line 1) is $O(n^2 + 1)$, and the broadcast operation in parallel for each MEC server (line 2) is $O(m)$. In each iteration, the time complexity of obtaining the optimal offloading server $\boldsymbol{O}_i^*$ for each server in parallel through Algorithm 2 (line 6) is $O(2^m \cdot (m + \sigma))$. Supposing the time complexity of obtaining the optimal offloading vector $X_i^{(k)}$ with CVX toolkit (line 7) is $O(\xi)$, and the maximum number of iterations is $L$. Accordingly, the time complexity of the iteration process (lines 3-12) is $O(L \cdot (2^m \cdot (m + \sigma) + \xi))$. Therefore, time complexity of SBOA is $O(L \cdot (2^m \cdot (m + \sigma) + \xi) + n^2 + m + 1)$, which can be simplified to $O(L \cdot (2^m \cdot (m + \sigma) + \xi) + n^2)$.

## VI. Performance Analysis

Massive experiments are conducted in this section to evaluate the performance of our proposed algorithm (i.e., SBOA). In particular, the following Research Questions (**RQs**) are discussed with the experimental evaluations.

*RQ1:* Can our proposed algorithm efficiently converge to a NE point, and reduce the average completion time of system? (Section VI-C)

*RQ2:* Is SBOA better than other benchmark algorithms in terms of reducing the average completion time of system after balancing offloading? (Section VI-D)

*RQ3:* Do different volumes of arriving tasks have an impact on the performance of SBOA? (Section VI-E)

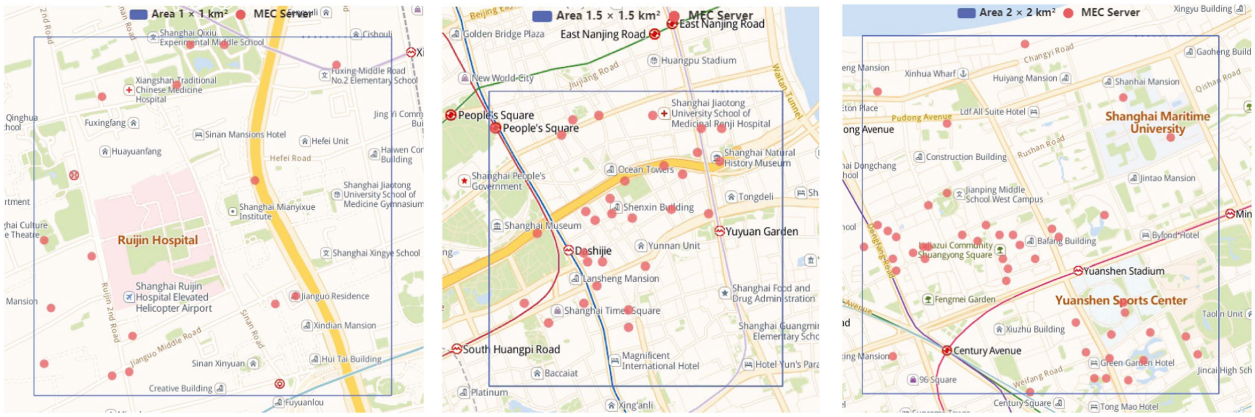
### A. Basic Experimental Setup

All simulation experiments are run on the Win11 64-bit operating system with an Intel(R) Core(TM) i5-10500 CPU at 3.10 GHz and 8 GB RAM. Both the proposed SBOA and other algorithms are implemented based on Python 3.10.

We build a multi-edge environment with the dataset of MEC server distribution with latitude and longitude information in Shanghai [48], [49], [50]. There are three different scale of MEC servers in our experiments, where $n = \{15, 30, 45\}$. Figs. 2(a), (b) and (c) display the actual location distribution of MEC servers with different scales (i.e., small, medium and large scale), respectively. The system model are constructed with Pandas 1.3 and Numpy 1.20, and CVXPY 1.2 and its solver CPLEX are used to solve the convex functions. In each scenario, the average task arrival rate $\lambda_i$ and the service rate $f_i$ follow the normal distribution $N(10, 4)$ and $N(15, 6)$, respectively [30]. According to the distance between MEC servers in Shanghai, the transmission time per unit of task volume $\boldsymbol{D}$ between MEC servers is mapping to the interval $[0.1, 0.2]s$ [51]. Moreover, the number of adjacent servers $m$ satisfies $0 < m \leq 3$, where $m \in \boldsymbol{Z}$ [24].

### B. Benchmark Algorithms

To analyze the performance of the proposed SBOA in reducing the average completion time of MEC servers, and its




(a) $n$=15 (small scale) (b) $n$=30 (medium scale) (c) $n$=45 (large scale)

Fig. 2. Actual location distribution of MEC servers with different server scales.

adaptability to different scenarios, we introduce the following four benchmark algorithms.

(**i**) LC (Local Computing): Each MEC server executes its tasks locally without computation offloading.

(**ii**) PSOGA (Particle Swarm Optimization algorithm employing the update operators of Genetic Algorithm) [30]: It is a centralized optimization algorithm inspired by bird predation behavior and Darwinian evolutionary theory [52], [53]. PSOGA constructs the computation offloading of MEC servers as a particle, and the approximate optimal solution of the optimization problem can be obtained through particle search and mutation [54]. The related parameters and particle update operations are set based on [28]. The number of iterations and the population size are set to 10000 and 1000, respectively.

(**iii**) IPA (Iterative Proximal Algorithm) [15]: It constructs a perfect-information and non-cooperative game in multi-edge environments, and iteratively solves the NE solution to obtain an optimal computation offloading strategy among MEC servers. When the difference between the results of the current iteration and the results of the previous iteration is less than $\varepsilon$, the NE point is considered to have been reached and the current solution is taken as the best one for the game. According to [15], the parameter $\varepsilon$ is set to 0.1.

(**iii**) DRL (Deep Reinforcement Learning) [55]: It combines deep neural networks with reinforcement learning technique, which employs the DRL method based on DQN. We use $\{\lambda_i, s_i\}$ as the state space, and use $\boldsymbol{X_i}$ as the action space. Both the target network and the policy network consist of 1 input layer, 2 hidden layers, and 1 output layer, where each hidden layer has 128 hidden neurons. The learning rate is set to 0.001, the discount factor is set to 0.99, the capacity of the replay buffer is set to 100000, and the batch size is set to 32 [51].

### C. *RQ1: Convergence and Effectiveness*

We try to verify the convergence of SBOA, thereby verifying whether our proposed scheme can effectively converges to a NE point. Fig. 3 displays the aggregation task arrival rate of $e_i$ (i.e., $s_i$) versus the number of iterations of SBOA. Fig. 4 depicts the average completion time of $e_i$ (i.e., $t_i^{off}$) versus

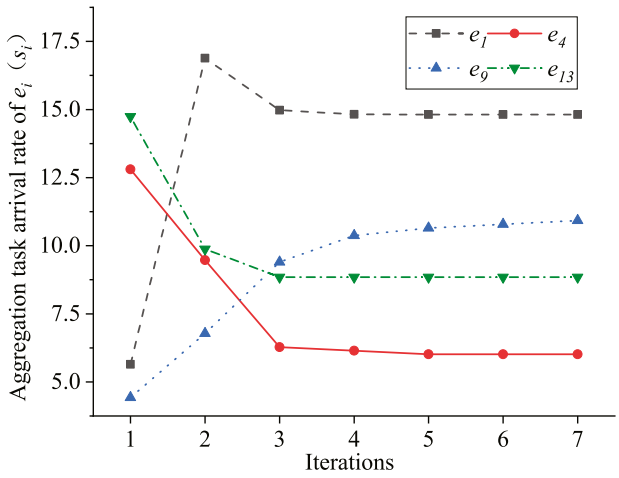



Fig. 3. Aggregation task arrival rate of $e_i$ versus the number of iterations of SBOA.

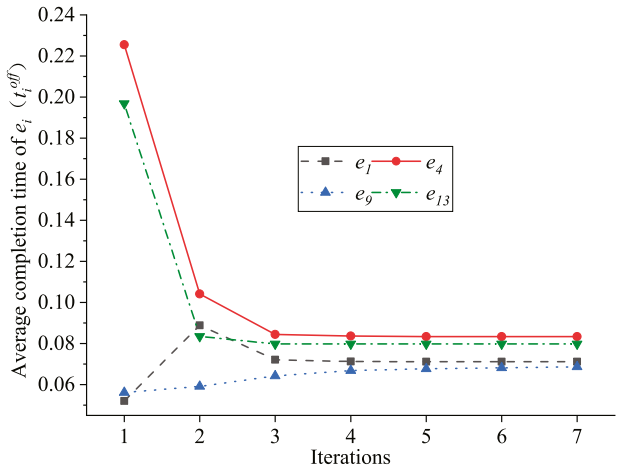



Fig. 4. Average completion time of $e_i$ versus the number of iterations of SBOA.

the number of iterations of SBOA. In these experiments, we randomly selected 4 servers (i.e., $e_1, e_4, e_9$, and $e_{13}$) from 15 MEC servers to observe how the aggregation task arrival rate and average completion time of $e_i$ change. In Fig. 3, the aggregation

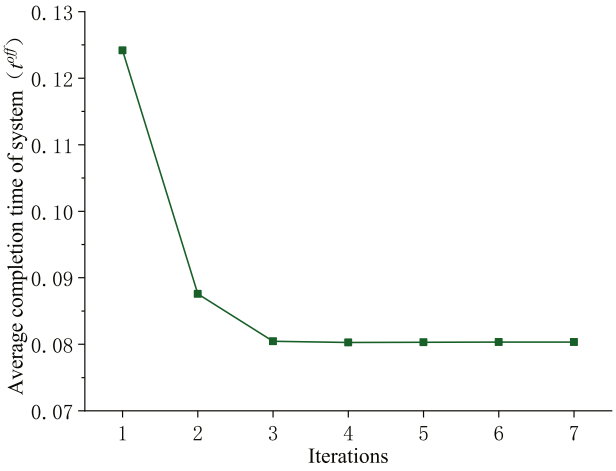



Fig. 5. Average completion time of system versus the number of iterations of SBOA.

task arrival rate of $e_1$ increases rapidly and then decreases. This is because that $e_1$ starts with a low-load state, and other servers will choose to offload tasks to it for execution, resulting in overload in $e_1$ upfront. As the number of iterations increases, $e_1$ reduces its own load by offloading its arrival tasks to other low-load adjacent servers. Correspondingly, the average completion time of $e_1$ increases rapidly and then decreases in Fig. 4. Both $e_4$ and $e_{13}$'s load decrease monotonically. They can obtain better computation offloading strategies in each iteration, and offload their tasks to suitable servers to reduce the average completion time. $e_9$'s load increase monotonically, which is in general the target for other servers' computation offloading in each iteration. From Figs. 3 and 4, we can see that both the aggregation task arrival rate and the average completion time of $e_i$ tend to be stable after a limited number of iterations. Therefore, SBOA can converge to a NE point in a finite number of iterations. Specifically, both $s_i$ and $t_{i,j}^{off}$ reach a relatively stable state after about 5 iterations, which indicates the high efficiency of SBOA.

Fig. 5 shows the average completion time of system versus the number of iterations of SBOA. The average completion time of system, $t^{off}$, is defined as follows.

$$t^{off} = \frac{\sum_{e_i \in \boldsymbol{E}} t_i^{off}}{n}. \tag{21}$$

Although the average completion time of $e_i$ does not always decrease during computation offloading in Fig. 4, the average completion time of system in Fig. 5 maintain a monotonically decreasing trend. After the system reaches a steady state, the average completion time of system drops by 35.3%, which indicates that SBOA has good balancing performance for reducing the computation pressure on high-load servers.

### D. *RQ2: Performance Evaluation and Comparisons*

This work focuses on reducing the average completion time of system, which maps to the average disutility of system $Q$.

$$Q = \frac{\sum_{e_i \in \boldsymbol{E}} Q_i}{n}. \tag{22}$$

Fig. 6 displays the average disutility of system with different service rates in various scenarios (i.e., small, medium, and large), where the service rate $f_i$ follows different normal distributions (i.e., $N(15,6)$, $N(13,5)$ and $N(11,4)$). From the overview in Fig. 6, we can obviously find that the average disutility of system shows a monotonically increasing trend as the service rate decreases. This is because that MEC servers have to handle the same volume of tasks with reduced service rate as the $f_i$ decreases, leading to a consequent rise in the average completion time of system. LC's performance is worst in all three scenarios. LC executes tasks locally without computation offloading, hence there is no synergistic computation offloading among MEC servers with LC.

Fig. 6(a) displays the average disutility of system with different service rates for the small-scale scenario. PSOGA has the best performance, which is mainly due to its global search nature, resulting in finding optimal solutions in the small-scale scenario. The maximum performance gap between SBOA and PSOGA is 3.7% and the average gap is only 1.6%. Compared with LC, IPA, and DRL, SBOA reduces the average disutility of system by 27.4%, 3.3%, and 4.2%, respectively. In Fig. 6(b), the performance gap between SBOA and PSOGA is $-1.6\%{-}4.0\%$ and the average gap is also 1.6%. As the solution space increases, PSOGA cannot achieve the optimal performance within a limited number of iterations. SBOA decreases the average disutility of system by 14.7% compared with IPA. The main reason for the large performance gap between IPA and SBOA is that IPA makes offloading decisions with perfect information, which increases the algorithm's decision space with more MEC servers, leading to falling into poorer local optimal solutions with a high probability. Fig. 6(c) displays the average disutility of system with different service rates for the large-scale scenario. SBOA has the best performance, which is 1.1%-7.1% better than that of PSOGA. PSOGA's performance is worse for the large-scale scenario. This is because that the crossover and mutation operations for update hardly improve particle quality for large-scale solution space. In summary, SBOA slightly underperforms PSOGA with a small number of MEC servers. However, as the number of servers increases, the performance of PSOGA gradually decreases, while SBOA is able to show superior performance. Notably, SBOA's stability and effectiveness are significantly better than that of IPA, DRL, and LC in either scenario.

Fig. 7 exhibits the average disutility of system with different task arrival rates in various scenarios (i.e., small, medium, and large), where the task arrival rate $\lambda_i$ follows different normal distributions (i.e., $N(10,4)$, $N(8,3)$ and $N(6,2)$). The average disutility of system monotonously declines as the task arrival rate $\lambda_i$ decreases. This is because that MEC servers only have to handle a smaller number of tasks with the same service rate $f_i$, resulting in a consequent decline for the average completion time of system.

Fig. 7(a) exhibits the average disutility of system with different task arrival rates for the small-scale scenario. PSOGA performs better than other algorithms, but the average performance gap between SBOA and PSOGA is only 0.5%. In Fig. 7(c), SBOA has the best performance, which is 2.2%, 2.8%, 3.7%,

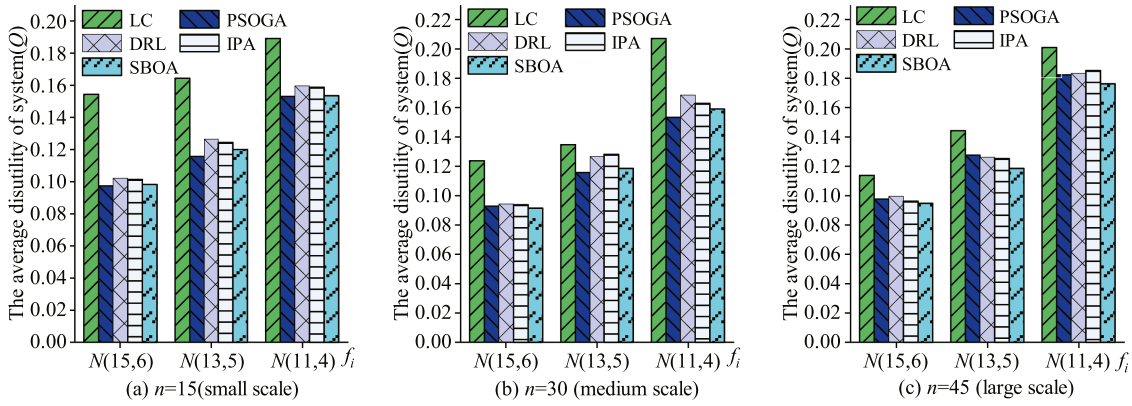



Fig. 6. Average disutility of system with different service rates in various scenarios.

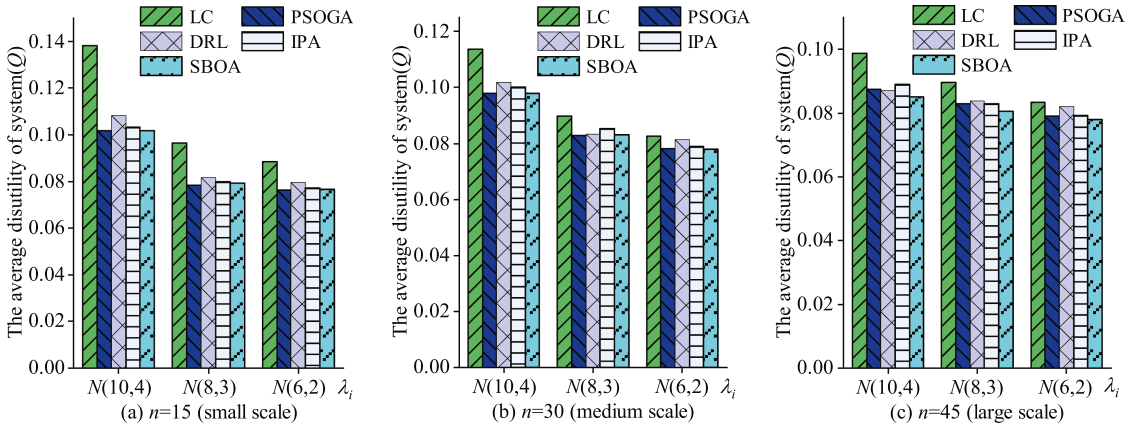



Fig. 7. Average disutility of system with different task arrival rates in various scenarios.

TABLE III
EXECUTION TIME COMPARISON WITH DIFFERENT SERVER SCALES (S)

| Server scale | SBOA | PSOGA | IPA | DRL |
|---|---|---|---|---|
| $n = 15$ | 1.20s | 348.2s | 1.31s | 0.90s |
| $n = 30$ | 1.37s | 1064.6s | 3.87s | 0.98s |
| $n = 45$ | 1.68s | 2131.2s | 7.10s | 1.21s |

and 10.0%, on average better than that of PSOGA, IPA, DRL, and LC, respectively.

Figs. 6 and 7 display that the performance of SBOA is better than that of DRL in all scenarios. This is because DRL relies on formed strategies during training and historical data, and generates results through prediction methods. In contrast, the proposed SBOA uses convex optimization methods to directly generate the optimal offloading vector for each server through mathematically precise calculations. SBOA does not rely on prediction methods but convex optimization theory, ensuring that a better solution can be found compared with DRL.

Table III displays the execution time comparison of different algorithms (i.e., SBOA, PSOGA, IPA, and DRL) for finding the optimal balancing offloading solution with different server scales (i.e., small, medium, and large). The experimental setup is the same as in Section VI-A. For the small-scale scenario, the execution time of SBOA, IPA, and DRL is 1.20 s, 1.31 s, and 0.90 s, respectively, and those algorithms can satisfy the system management. PSOGA's execution time is 348.2 s, which makes it difficult to adapt to the multi-edge environments with dynamically changing load. As the number of MEC servers increases to 30 and 45, the execution time of IPA increases to 3.87 s and 7.10 s, respectively. In contrast, the execution time of SBOA only increases to 1.37 s and 1.68 s, which does not increase abruptly with the number of servers. This is because IPA takes more time to make offloading decisions when faced with a larger solution space with perfect information (i.e., all servers' information), whereas SBOA can quickly make offloading decisions when faced with a small solution space with imperfect information (i.e., adjacent servers' information). Therefore, SBOA can effectively deal with large-scale scenarios. The execution time of DRL is slightly better than that of SBOA, with 0.98 s and 1.21 s in medium-scale and large-scale scenarios, respectively. DRL, like SBOA, also generates computation offloading schemes with imperfect information through distributed decision-making manner. It trains models on historical data and obtains results quickly through prediction methods. Table III shows that the

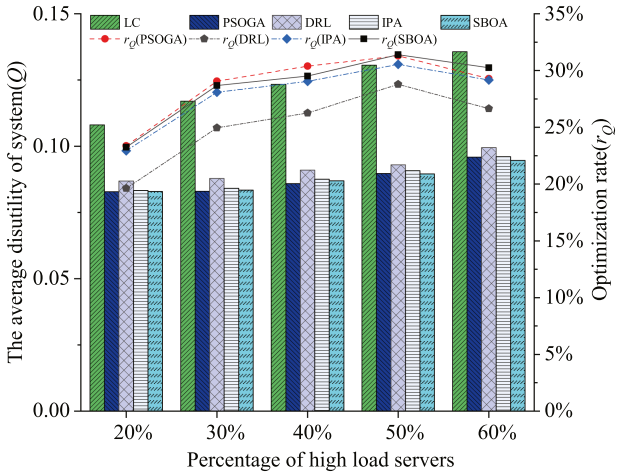



Fig. 8. Algorithms' performance with different volumes of arriving tasks.

execution time of DRL is better than that of SBOA in three scenarios, but they are of the same order of magnitude.

### E. *RQ3: Impact of Task Volume*

To test the effects of different volumes of arriving tasks on the performance of SBOA, we conduct the following experiments in the medium-scale scenario. The 30 MEC servers are categorized into 2 groups: one with high task volume, and the other with low task volume. The task arrival rate $\lambda_i$ for the high-load servers follows the normal distribution $N(14, 4)$, and the low-load servers follows the normal distribution $N(5, 4)$. To demonstrate the superiority of the computation offloading method, we introduce the optimization rate (i.e., $r_Q$) as an evaluation metric.

$$r_Q(M) = \frac{Q(\mathrm{LC}) - Q(M)}{Q(\mathrm{LC})}, \tag{23}$$

where LC is one of the benchmark algorithms discussed in Section VI-B, $M$ is one of the algorithms (i.e., PSOGA, IPA, DRL, and SBOA) except LC, and $Q(M)$ corresponds to the average disutility of system with the corresponding algorithm $M$.

Fig. 8 displays the algorithms' performance with different volumes of arriving tasks. There are 5 groups of experiments, where the percentage of high-load servers is {20%, 30%, 40%, 50%, 60%}, and the rest are low-load servers. When the percentage of high-load servers is 20%, the optimization rate of SBOA, $r_Q(\mathrm{SBOA})$, is lower (i.e., only 23.2%). This is because there are fewer tasks with high response time that can be optimized by computation offloading when the percentage of high-load servers is less. When the percentage of high-load servers increases to 30%, 40%, and 50%, the $r_Q(\mathrm{SBOA})$ gradually increases to 28.7%, 29.5%, and 31.4%, respectively. The reason for this is that as the number of high-load server increases, the number of high responsive task to be optimized also increases, and the optimization rate is improved by offloading these tasks to other idle servers. When the percentage of high-load servers is 60%, the $r_Q(\mathrm{SBOA})$ slightly decreases to 30.3%. This is because although there are more optimizable tasks, there is a lack of enough idle servers to offload these tasks to limit the optimization rate. Note that the optimization rate of SBOA is better than that of other algorithms when the percentage of high-load servers is more than 50%. SBOA and PSOGA outperform each other in the medium-scale scenario, which is demonstrated in ***RQ2***. As the percentage of high-load servers rises and the number of optimizable tasks increases, PSOGA face greater challenges in finding the ideal balancing offloading solution. In contrast, SBOA shows more adaptability in these complex scenarios, and therefore obtains better performance. Overall, the average $r_Q(\mathrm{SBOA})$ in these scenarios is 28.6%, which indicates that SBOA can maintain better performance of computation offloading with different volumes of arriving tasks.

## VII. Conclusion and the Future Work

This paper has proposed a Synergistic Balancing Offloading Algorithm (SBOA) through distributed decision-making manner to obtain the optimal offloading strategy in multi-edge environments. The synergistic computation offloading among MEC servers is modeled as a non-cooperative game with imperfect information. Furthermore, the game is mathematically analyzed to prove that there is a Nash Equilibrium (NE) point. Simulation results show that SBOA has good balancing performance for reducing the computation pressure on high-load servers, and its stability and effectiveness are significantly better than that of IPA, DRL, and LC in either scenario. Particularly, SBOA can effectively reduce the system completion time in large-scale scenarios.

Our future work will handle the computation offloading for dependent tasks in multi-edge environments. Simultaneously, we will further consider the energy consumption of MDs, and try to maximize their energy efficiency.

## References

[1] Y. Liu, M. Peng, G. Shou, Y. Chen, and S. Chen, "Toward edge intelligence: Multiaccess edge computing for 5G and Internet of Things," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 6722–6747, Aug. 2020.
[2] J. Mei, Z. Tong, K. Li, L. Zhang, and K. Li, "Energy-efficient heuristic computation offloading with delay constraints in mobile edge computing," *IEEE Trans. Serv. Comput.*, vol. 16, no. 6, pp. 4404–4417, Nov./Dec. 2023.
[3] Y. Yang, H. Shen, and H. Tian, "Scheduling workflow tasks with unknown task execution time by combining machine-learning and greedy-optimization," *IEEE Trans. Serv. Comput.*, vol. 17, no. 3, pp. 1181–1195, May/Jun. 2024.
[4] Y. Wang, I. R. Chen, and D. C. Wang, "A survey of mobile cloud computing applications: Perspectives and challenges," *Wireless Pers. Commun.*, vol. 80, pp. 1607–1623, 2015.
[5] A. U. R. Khan, M. Othman, S. A. Madani, and S. U. Khan, "A survey of mobile cloud computing application models," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 1, pp. 393–413, First Quarter, 2014.
[6] L. Luo, H. Yu, K.-T. Foerster, M. Noormohammadpour, and S. Schmid, "Inter-datacenter bulk transfers: Trends and challenges," *IEEE Netw.*, vol. 34, no. 5, pp. 240–246, Sep./Oct. 2020.
[7] S. Chu, C. Gao, M. Xu, K. Ye, Z. Xiao, and C. Xu, "Efficient multi-task computation offloading game for mobile edge computing," *IEEE Trans. Serv. Comput.*, vol. 17, no. 1, pp. 30–46, Jan./Feb. 2024.
[8] X. Xu et al., "An energy-aware computation offloading method for smart edge computing in wireless metropolitan area networks," *J. Netw. Comput. Appl.*, vol. 133, pp. 75–85, 2019.

[9] S. Wang et al., "Adaptive federated learning in resource constrained edge computing systems," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1205–1221, Jun. 2019.

[10] X. Xu et al., "Dynamic resource allocation for load balancing in fog environment," *Wireless Commun. Mobile Comput.*, vol. 50, no. 10, pp. 58–67, 2018.

[11] J. Tang, T. Qin, Y. Xiang, Z. Zhou, and J. Gu, "Optimization search strategy for task offloading from collaborative edge computing," *IEEE Trans. Serv. Comput.*, vol. 16, no. 3, pp. 2044–2058, May/Jun. 2023.

[12] Q. Liu, T. Xia, L. Cheng, M. Van Eijk, T. Ozcelebi, and Y. Mao, "Deep reinforcement learning for load-balancing aware network control in IoT edge systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 6, pp. 1491–1502, Jun. 2022.

[13] J. Li et al., "An end-to-end load balancer based on deep learning for vehicular network traffic control," *IEEE Internet Things J.*, vol. 6, no. 1, pp. 953–966, Feb. 2019.

[14] C. Yang, X. Xu, M. Bilal, Y. Wen, and T. Huang, "Deep-deterministic-policy-gradient-based task offloading with optimized k-means in edge-computing-enabled IoMT cyber-physical systems," *IEEE Syst. J.*, vol. 17, no. 4, pp. 5195–5206, Dec. 2023.

[15] C. Liu, K. Li, and K. Li, "A game approach to multi-servers load balancing with load-dependent server availability consideration," *IEEE Trans. Cloud Comput.*, vol. 9, no. 1, pp. 1–13, First Quarter, 2021.

[16] W. Fan, L. Yao, J. Han, F. Wu, and Y. Liu, "Game-based multitype task offloading among mobile-edge-computing-enabled base stations," *IEEE Internet Things J.*, vol. 8, no. 24, pp. 17691–17704, Dec. 2021.

[17] S. Duan et al., "MOTO: Mobility-aware online task offloading with adaptive load balancing in small-cell MEC," *IEEE Trans. Mobile Comput.*, vol. 23, no. 1, pp. 645–659, Jan. 2024.

[18] Y. Xu, T. Mohammed, M. Di Francesco, and C. Fischione, "Distributed assignment with load balancing for DNN inference at the edge," *IEEE Internet Things J.*, vol. 10, no. 2, pp. 1053–1065, Jan. 2023.

[19] L. Li, M. Siew, Z. Chen, and T. Q. Quek, "Optimal pricing for job offloading in the MEC system with two priority classes," *IEEE Trans. Veh. Technol.*, vol. 70, no. 8, pp. 8080–8091, Aug. 2021.

[20] M.-H. Chen, B. Liang, and M. Dong, "Joint offloading and resource allocation for computation and communication in mobile cloud with computing access point," in *Proc. IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.

[21] 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.

[22] H. Zhou, Z. Wang, G. Min, and H. Zhang, "UAV-aided computation offloading in mobile-edge computing networks: A stackelberg game approach," *IEEE Internet Things J.*, vol. 10, no. 8, pp. 6622–6633, Apr. 2023.

[23] S. Liu, Y. Yu, L. Guo, P. L. Yeoh, B. Vucetic, and Y. Li, "Adaptive delay-energy balanced partial offloading strategy in mobile edge computing networks," *Digit. Commun. Netw.*, vol. 9, no. 6, pp. 1310–1318, 2023.

[24] X. Zhu, T. Zhang, J. Zhang, B. Zhao, S. Zhang, and C. Wu, "Deep reinforcement learning-based edge computing offloading algorithm for software-defined IoT," *Comput. Netw.*, vol. 235, 2023, Art. no. 110006.

[25] Y. Chen, W. Gu, J. Xu, Y. Zhang, and G. Min, "Dynamic task offloading for digital twin-empowered mobile edge computing via deep reinforcement learning," *China Commun.*, vol. 20, no. 11, pp. 164–175, 2023.

[26] M. Diamanti, C. Pelekis, E. E. Tsiropoulou, and S. Papavassiliou, "Delay minimization for rate-splitting multiple access-based multi-server MEC offloading," *IEEE/ACM Trans. Netw.*, vol. 32, no. 2, pp. 1035–1047, Apr. 2024.

[27] C. Yi, J. Cai, T. Zhang, K. Zhu, B. Chen, and Q. Wu, "Workload reallocation for edge computing with server collaboration: A cooperative queueing game approach," *IEEE Trans. Mobile Comput.*, vol. 22, no. 5, pp. 3095–3111, May 2023.

[28] L. Yang, H. Yao, J. Wang, C. Jiang, A. Benslimane, and Y. Liu, "Multi-UAV-enabled load-balance mobile-edge computing for IoT networks," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 6898–6908, Aug. 2020.

[29] J. Wan, B. Chen, S. Wang, M. Xia, D. Li, and C. Liu, "Fog computing for energy-aware load balancing and scheduling in smart factory," *IEEE Trans. Ind. Inform.*, vol. 14, no. 10, pp. 4548–4556, Oct. 2018.

[30] B. Lin et al., "A time-driven data placement strategy for a scientific workflow combining edge computing and cloud computing," *IEEE Trans. Ind. Inform.*, vol. 15, no. 7, pp. 4254–4265, Jul. 2019.

[31] J. Tan, R. Khalili, H. Karl, and A. Hecker, "Multi-agent distributed reinforcement learning for making decentralized offloading decisions," in *Proc. 2022 IEEE Conf. Comput. Commun.*, 2022, pp. 2098–2107.

[32] M. Hu, Z. Xie, D. Wu, Y. Zhou, X. Chen, and L. Xiao, "Heterogeneous edge offloading with incomplete information: A minority game approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 9, pp. 2139–2154, Sep. 2020.

[33] Q. Wang, Y. Zhu, and X. Wang, "Incomplete information based collaborative computing in emergency communication networks," *IEEE Commun. Lett.*, vol. 24, no. 9, pp. 2038–2042, Sep. 2020.

[34] A. M. Jasim and H. Al-Raweshidy, "An adaptive SDN-based load balancing method for edge/fog-based real-time healthcare systems," *IEEE Syst. J.*, vol. 18, no. 2, pp. 1139–1150, Jun. 2024.

[35] X. Chen, J. Hu, Z. Chen, B. Lin, N. Xiong, and G. Min, "A reinforcement learning-empowered feedback control system for industrial Internet of Things," *IEEE Trans. Ind. Inform.*, vol. 18, no. 4, pp. 2724–2733, Apr. 2022.

[36] M. Jia, W. Liang, Z. Xu, M. Huang, and Y. Ma, "QoS-aware cloudlet load balancing in wireless metropolitan area networks," *IEEE Trans. Cloud Comput.*, vol. 8, no. 2, pp. 623–634, Second Quarter 2020.

[37] J. F. Shortle, J. M. Thompson, D. Gross, and C. M. Harris, *Fundamentals of Queueing Theory*, vol. 399. Hoboken, NJ, USA: Wiley, 2018.

[38] G. Li, J. Cai, X. Chen, and Z. Su, "Nonlinear online incentive mechanism design in edge computing systems with energy budget," *IEEE Trans. Mobile Comput.*, vol. 22, no. 7, pp. 4086–4102, Jul. 2023.

[39] G. Owen, *Game Theory*. Bingley, U.K.: Emerald Group Publishing, 2013.

[40] W. Dai, H. Lu, J. Xiao, and Z. Zheng, "Task allocation without communication based on incomplete information game theory for multi-robot systems," *J. Intell. Robotic Syst.*, vol. 94, pp. 841–856, 2019.

[41] J. Bhandari and D. Russo, "Global optimality guarantees for policy gradient methods," *Operations Res.*, vol. 72, no. 5, pp. 1906–1927, Sep. 2024.

[42] S. P. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.

[43] G. Scutari, D. P. Palomar, F. Facchinei, and J.-S. Pang, "Monotone games for cognitive radio systems," *Distrib. Decis. Mak. Control*, vol. 417, no. 1, pp. 83–112, 2012.

[44] Y.-H. Wu, Z.-C. Yu, C.-Y. Li, M.-J. He, B. Hua, and Z.-M. Chen, "Reinforcement learning in dual-arm trajectory planning for a free-floating space robot," *Aerosp. Sci. Technol.*, vol. 98, 2020, Art. no. 105657.

[45] A. Laha, B. Yin, Y. Cheng, L. X. Cai, and Y. Wang, "Game theory based charging solution for networked electric vehicles: A location-aware approach," *IEEE Trans. Veh. Technol.*, vol. 68, no. 7, pp. 6352–6364, Jul. 2019.

[46] M. De Santis, G. Eichfelder, J. Niebling, and S. Rocktäschel, "Solving multiobjective mixed integer convex optimization problems," *SIAM J. Optim.*, vol. 30, no. 4, pp. 3122–3145, 2020, doi: 10.1137/19M1264709.

[47] Z. Zhao, S. Liu, M. Zhou, D. You, and X. Guo, "Heuristic scheduling of batch production processes based on petri nets and iterated greedy algorithms," *IEEE Trans. Automat. Sci. Eng.*, vol. 19, no. 1, pp. 251–261, Jan. 2022.

[48] Y. Li, A. Zhou, X. Ma, and S. Wang, "Profit-aware edge server placement," *IEEE Internet Things J.*, vol. 9, no. 1, pp. 55–67, Jan. 2022.

[49] Y. Guo, S. Wang, A. Zhou, J. Xu, J. Yuan, and C.-H. Hsu, "User allocation-aware edge cloud placement in mobile edge computing," *Softw.: Pract. Exp.*, vol. 50, no. 5, pp. 489–502, 2020.

[50] S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou, and X. Shen, "Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach," *IEEE Trans. Mobile Comput.*, vol. 20, no. 3, pp. 939–951, Mar. 2021.

[51] X. Chen, Z. Yao, Z. Chen, G. Min, X. Zheng, and C. Rong, "Load balancing for multiedge collaboration in wireless metropolitan area networks: A two-stage decision-making approach," *IEEE Internet Things J.*, vol. 10, no. 19, pp. 17124–17136, Oct. 2023.

[52] Z. Xiao, Q. Qiu, L. Li, Y. Feng, Q. Lin, and Z. Ming, "An efficient service-aware virtual machine scheduling approach based on multi-objective evolutionary algorithm," *IEEE Trans. Serv. Comput.*, vol. 17, no. 5, pp. 2027–2040, Sep./Oct. 2024.

[53] T. Bäck and H.-P. Schwefel, "An overview of evolutionary algorithms for parameter optimization," *Evol. Comput.*, vol. 1, no. 1, pp. 1–23, 1993.

[54] F. Xu, Z. Zhang, J. Feng, Z. Qin, and Y. Xie, "Efficient deployment of multi-UAV assisted mobile edge computing: A cost and energy perspective," *Trans. Emerg. Telecommun. Technol.*, vol. 33, no. 5, 2022, Art. no. e4453.

[55] H. Cao and J. Cai, "Distributed multiuser computation offloading for cloudlet-based mobile cloud computing: A game-theoretic machine learning approach," *IEEE Trans. Veh. Technol.*, vol. 67, no. 1, pp. 752–764, Jan. 2018.

**Bing Lin** received the BS and MS degrees in computer science from Fuzhou University, Fuzhou, China, in 2010 and 2013, respectively, and the PhD degree in communication and information system from Fuzhou University, in 2016. He is currently an associate professor with the College of Physics and Energy, Fujian Normal University. Now he is the deputy director of the Department of Energy and Materials, and leads the Intelligent Computing research group. His research interest mainly includes parallel and distributed computing, computational intelligence, and data center resource management. He has published more than fifty journals and conference articles, such as *Transactions on Parallel and Distributed Systems*, *Transactions on Industrial Informatics*, *Transactions on Network and Service Management*, and *IoT J.*, etc.

**Jie Weng** received the BS degree in computer science and technology from Fujian Normal University, Fujian, China, in 2021. He is currently working toward the MS degree in software engineering with the College of Computer and Data Science, Fuzhou University, Fuzhou, China. Since September 2021, he has also been a part of Fujian Key Laboratory of Network Computing and Intelligent Information Processing, Fuzhou University. His main research interests include game theory, mobile edge computing and load balancing.

**Xing Chen** (Member, IEEE) is a professor with Fuzhou University, and the director of Fujian Key Laboratory of Network Computing and Intelligent Information Processing. He focuses on the software systems and engineering approaches for cloud and mobility. His current projects cover the topics from self-adaptive software, computation offloading, model driven approach and so on. He has published more than 80 journal and conference articles, including *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Industrial Informatics*, etc.

**Yun Ma** received the PhD degree majoring in computer science from the School of EECS, Peking University, under the direction of Professor Hong Mei and Professor Gang Huang. His research interests lie in mobile computing, Web technologies, and services computing Currently, he focuses on synergy between the mobile and the Web, trying to improve the mobile user experience by leveraging the best practices from native apps and Web apps.

**Ching-Hsien Hsu** (Senior Member, IEEE) is currently a chair professor and the dean of the College of Information and Electrical Engineering, Asia University, Taiwan, and a professor with the Department of Computer Science and Information Engineering, National Chung Cheng University. His research interests include high performance computing, cloud computing, parallel and distributed systems, Big Data analytics, and ubiquitous/pervasive computing and intelligence. He has published 200 papers in top journals such as *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Services Computing*, *ACM Transactions on Multimedia Computing, Communications, and Applications*, *IEEE Transactions on Cloud Computing*, *IEEE Transactions on Emerging Topics in Computing*, *IEEE System*, *IEEE Network*, top conference proceedings, and book chapters in these areas. He has been acting as an author/coauthor or an editor/co-editor of ten books from Elsevier, Springer, IGI Global, World Scientific, and McGraw-Hill. He is a fellow of the Institution of Engineering and Technology.