AoI-Oriented Computation Offloading and Resource Allocation for End–Edge–Cloud Computing Systems

Youling Zeng, Yue Zeng, Jining Chen, Yufan Shen, Liying Li, Peijin Cong, Junlong Zhou, *Member, IEEE*, and Keqin Li, *Fellow, IEEE*

***Abstract*—As smart mobile applications increasingly demand timely situational awareness and energy efficiency, the Age of Information (AoI) metric plays a vital role in maintaining data freshness. This need is further supported by the end–edge–cloud computing (EECC) paradigm, which enhances application performance by facilitating task offloading to the edge or the cloud. However, existing AoI optimization solutions focus solely on task offloading, often neglecting critical aspects, such as system resource allocation and energy efficiency, which can lead to resource waste, increased energy consumption, compromised Quality of Service (QoS), and system performance degradation. Therefore, this article investigates the joint optimization of task offloading, communication and computing resource allocation in EECC systems, aiming to minimize AoI and energy consumption under constraints of deadlines and capacity constraints. To address this problem, we divide the decision space into multiple nonintersecting decision areas based on the characteristics of the studied problem and design a task offloading and resource allocation algorithm based on slow-movement particle swarm optimization (SPSO) to handle each decision area individually. In the algorithm design, we customize the position, velocity, update rules, and fitness function for the optimization problem. Finally, extensive simulation-based and testbed experiment results show that the proposed algorithm can save up to 14.56% of energy consumption, shorten AoI by up to 27.80%, and improve utility (weighted sum of AoI and energy consumption) by up to 15.89% compared with existing algorithms.**

***Index Terms*—Age of Information (AoI), computation offloading, end–edge–cloud computing (EECC), resource allocation.**

Received 2 April 2025; revised 24 May 2025 and 1 July 2025; accepted 19 July 2025. Date of publication 22 July 2025; date of current version 25 September 2025. This work was supported in part by the National Natural Science Foundation of China under Grant 62302221, Grant 62302216, Grant 62172224, and Grant 62402226; in part by the Natural Science Foundation of Jiangsu Province under Grant BK20230913, Grant BK20230912, Grant BK20220138, and Grant BK20241453; in part by the Fundamental Research Funds for the Central Universities under Grant 30922010318, Grant 30924010815, and Grant 30924010817; and in part by the Open Project Program of Guangxi Key Laboratory of Digital Infrastructure under Grant GXDIOP2024006. *(Corresponding authors: Peijin Cong; Junlong Zhou.)*

Youling Zeng, Yufan Shen, Liying Li, Peijin Cong, and Junlong Zhou are with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China (e-mail: jlzhou@njust.edu.cn).

Yue Zeng is with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China, and also with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China.

Jining Chen is with the Digital Infrastructure Key Laboratory, Guangxi Zhuang Autonomous Region Information Center, Nanning 530022, China.

Keqin Li is with the Department of Computer Science, State University of New York, New Paltz, NY 12561 USA.

Digital Object Identifier 10.1109/JIOT.2025.3591682

I. Introduction

THE RAPID development of the Internet and smart mobile devices has led to a substantial increase in the number of users. According to a report by GSMA [1], the global mobile Internet user base reached 4.6 billion by the end of 2023. This growth has not only resulted in an explosion of data but also spurred the emergence of numerous latency-sensitive and computation-intensive applications, such as virtual reality, camera tracker, health management, and online gaming. A crucial aspect of these applications is the need for timely situational awareness. Ensuring decisions are made promptly and accurately requires up-to-date information, as outdated data can interfere with decision-making and lead to unforeseen losses. Traditional latency metrics, such as latency and throughput, do not fully capture information timeliness. The introduction of Age of Information (AoI) as a metric offers a more accurate measure of information freshness [2], [3]. AoI is defined as the time elapsed since the latest received data packet was generated at its source [4]. Unlike conventional metrics, AoI provides a perspective on information freshness from the receiver's perspective [5]. This metric addresses the quantitative needs of emerging applications for timely information and has rapidly become a key indicator for evaluating system performance across various domains.

As demands for Quality of Service (QoS) and energy efficiency continue to rise, maintaining efficient operation of applications on infrastructure has become increasingly complex [6]. To address these challenges, cloud computing (CC) and edge computing (EC) offer distinct paradigms for task processing. CC offloads tasks to remote cloud servers, alleviating local constraints but incurring high latency and network congestion [7]. Conversely, EC mitigates these issues by placing computing resources closer to users [8], but it suffers from resource limitations [9]. Multiaccess-edge computing (MEC)[1] [10] further enhances EC by deploying resources at the edge of mobile networks [11], [12], supporting timely

[1]Although MEC and end-edge CC (EECC) are closely related, their architectural scopes are fundamentally different. MEC, standardized by ETSI, primarily operates within a two-tier structure (end and edge), focusing on localized edge processing without explicitly modeling coordination with the cloud or end devices (EDs). In contrast, EECC introduces an explicit three-tier hierarchy encompassing the end, edge, and cloud layers, thereby supporting global task coordination and fine-grained resource management across all levels [13]. This distinction makes EECC a more suitable framework for system-wide optimization in increasingly complex and heterogeneous computing environments.

services in dynamic environments. However, as applications grow increasingly heterogeneous and system-scale coordination becomes critical, there is a need for a more comprehensive architectural model. EECC has thus emerged as a unified three-tier framework that enables collaborative task processing across end, edge, and cloud layers [13], offering greater flexibility and scalability. Within this framework, supporting timely and AoI-critical applications requires intelligent task offloading and adaptive resource allocation. Offloading refers to dynamically selecting the most appropriate execution layer—end, edge, or cloud—while resource allocation refers the efficient assignment of bandwidth and computing resources to ensure efficient and timely task execution.[2]

However, efficient computation offloading and resource allocation for AoI-oriented applications in EECC environments face the following challenges. First, in EECC systems where multiple users share resources, resource competition among mobile devices is inevitable. Thus, balancing computation offloading and resource allocation across these devices is critical. Second, variations in communication and computation overheads, resulting from different strategies, significantly impact the effectiveness of EECC systems. Third, the complexity of the system introduces additional challenges. The inherent resource heterogeneity necessitates decision-making that accounts for differences among various nodes, while latency sensitivity demands that tasks be processed promptly to meet deadlines. Lastly, the multitiered architecture of EECC adds another layer of complexity related to cross-layer coordination and optimization. Furthermore, energy management complicates the situation by requiring a balance between the limited energy of user devices and performance needs to extend their lifespan.

To address the aforementioned challenges, existing research has examined various approaches. Conventional studies [6], [7], [14], [15], [16] primarily focus on optimizing either latency or energy. While latency-oriented strategies can reduce task completion time, they often fail to ensure the freshness and timeliness of information, which are critical for decision-making in dynamic environments. This shortcoming can limit the efficacy of these strategies for real-time applications. Besides, recent research [3], [4], [5], [17], [18], [19], [20], [21], [22] has investigated AoI-oriented task scheduling and computation offloading, aiming to minimize peak or average AoI to guide strategy selection. Notably, metaheuristic algorithms [23], [24] have demonstrated potential in handling AoI optimization through multiobjective balancing. However, existing AoI-oriented studies on offloading or scheduling often lack a holistic consideration of energy efficiency, communication resource allocation, or computing resource allocation, with most works addressing only a subset of these aspects. This partial optimization may lead to inefficient resource utilization, increased energy consumption, and degraded overall system performance.

In this article, we investigate the joint task offloading and resource allocation problem aimed at minimizing AoI and energy consumption in EECC environments constrained by deadlines, computation, and bandwidth capacity. The former decides the execution location, while the latter allocates resources to enable efficient execution. Specifically, we first derive the average AoI formula for tasks on mobile devices and formalize the optimization problem as a mixed integer nonlinear programming (MINLP) problem. To address this problem, we then divide its decision space into multiple nonintersecting decision areas based on its characteristics and process each decision area individually. Afterwards, we propose a two-phase computation offloading and resource allocation algorithm based on the slow-movement particle swarm optimization (SPSO) algorithm, where the position vectors, velocity vectors, update rules, and fitness function in SPSO are customized for our studied problem. Finally, we build a real testbed platform and conduct extensive simulation and testbed-based experiments to verify its superiority over existing solutions in terms of AoI and energy consumption. Our main contributions are listed as follows.

1) To the best of our knowledge, this is the first work that jointly optimizes task offloading, communication and computing resource allocation in an EECC system, while minimizing both AoI and energy under deadline and capacity constraints. We formalize the problem as a MINLP problem and analyze its complexity.
2) We divide the decision space into multiple nonintersecting decision areas based on the characteristics of the studied problem and design an SPSO-based computation offloading and resource allocation algorithm to handle each decision area individually, where the position, velocity, update rules, and fitness function are customized for the problem.
3) We validate the efficacy of our approach through extensive simulation and testbed-based experiments. Evaluation results show that, compared with existing solutions, our approach can reduce average AoI by up to 27.80%, save energy consumption by up to 14.56%, and improve utility by up to 15.89%.

The rest of this article is outlined as follows. Section II reviews the related work. Section III presents the architecture and relevant models. Section IV deduces the average AoI calculation formula and defines the studied problem. The proposed algorithm is elaborated in Section V, and is evaluated in Section VI. Section VII concludes this article.

II. Related Work

This section classifies the related work into two categories: conventional metric-oriented optimization and AoI-oriented optimization. A comparative summary of these works, in terms of their optimization objectives and decision variables, is provided in Table I.

[2]In this work, task offloading refers to the decision regarding where a task should be executed—on the ED, at the edge server (ES), or in the cloud—based on latency, energy, and workload considerations. Resource allocation refers to how communication and computing resources (e.g., bandwidth and CPU cycles) are distributed to the selected execution nodes to support efficient task processing. Meanwhile, task scheduling, which involves determining the execution order or timing of tasks on a specific node to manage queueing or meet deadlines, is not included in our system model and is beyond the scope of this work.

TABLE I
COMPARISON OF OPTIMIZATION OBJECTIVES AND DECISION VARIABLES IN EXISTING STUDIES AND THIS WORK

Ref.	Optimization Objectives		Decision Variables		
				Resource Allocation	
	AoI	Energy	Offloading	Communication	Computing
[14], [25]	×	✓	✓	×	×
[15]	×	✓	✓	✓	✓
[7], [27]	×	×	✓	✓	✓
[26]	×	×	✓	✓	×
[28]	×	✓	✓	✓	×
[29]	×	✓	✓	×	✓
[30], [31]	×	✓	✓	×	×
[6]	×	✓	✓	✓	✓
[3], [4], [18]	✓	×	✓	×	×
[2], [17], [19], [33]	✓	×	×	×	×
[5]	✓	×	×	✓	×
[20], [21]	✓	✓	✓	×	×
[22]	✓	✓	✓	×	✓
[32]	×	✓	✓	✓	×
Our work	✓	✓	✓	✓	✓

A. *Conventional Metric-Oriented Optimization*

Existing research on EECC systems primarily addresses either energy consumption [14], [15], [25] or latency optimization issues [7], [26], [27]. For energy-focused offloading, [14] investigates energy-efficient strategies under deadline constraints, aiming to optimize system-level energy consumption. In contrast, Xiao et al. [15] and Zhai et al. [25] concentrated on device-level energy optimization. For example, Xiao et al. [15] explored how to allocate transmission power, bandwidth, and computing resources to minimize device energy consumption. Regarding latency, Qu and Wang [26] focused on emergency task offloading in smart factories, emphasizing the reduction of task execution delays. Kai et al. [7] and Liu et al. [27] included transmission power allocation, with Kai et al. optimizing total system latency and Liu et al. targeting average task latency. Some studies address both energy consumption and latency issues to enhance system performance. Tang et al. [28] used deep reinforcement learning to simultaneously improve average latency and energy consumption. Focusing on energy-delay tradeoffs, Zhai et al. [29] jointly optimized task offloading and computing resource allocation using deep Q-networks. Ding et al. [6] and Niu et al. [30] modeled the minimization of the weighted sum of latency and energy consumption as a multiuser computation offloading game. Although these studies effectively optimize task processing latency and energy consumption, they often neglect information freshness, which is crucial for emerging smart device applications. Outdated information can lead to erroneous decisions, diminishing system accuracy and reliability, and posing security threats.

B. *AoI-Oriented Optimization*

Recently, AoI has garnered significant attention in wireless and networked computing systems [2], [3], [4], [5], [17], [18], [19], [20], [21], [22], [31], [32], [33]. AoI measures the time elapsed since the most recent packet was generated, with factors like update frequency and processing time influencing its value. Reducing AoI can lead to lower response times and improved throughput [18]. Han et al. [2] proposed a physical-layer forwarding strategy for minimizing average AoI in multihop networks. Xu et al. [17] focused on minimizing peak AoI in autonomous driving using a reinforcement learning approach, while Song et al. [4] introduced the concept of Age of Task (AoT) with the goal of minimizing its sum, and Li et al. [18] proposed the concept of Age of Processing (AoP) and optimized the long-term average AoP by considering offloading strategies and sampling frequencies. Ndikumana et al. [3] considered an AoP-aware offloading strategy for autonomous vehicles, integrating a mobility-aware communication model with edge collaboration, but they lacked fine-grained, task-level resource optimization. Moreover, energy efficiency is usually ignored in the above approaches. Ma et al. [31] and Huang et al. [32] incorporated energy efficiency but only used AoI as a stability constraint. Studies by [5], [19], and [33] explore AoI-based data transmission and charging in wireless powered networks, focusing on charging efficiency rather than overall energy utilization. Song et al. [20] and Yang et al. [21] considered UAV-assisted systems and jointly optimized AoI and energy through trajectory planning and offloading decisions, yet they did not account for bandwidth or computing resource allocation. Wang et al. [22] maximized transmission energy efficiency under AoI constraints by optimizing signaling and beamforming, but did not consider task offloading and computing resource allocation. Meanwhile, metaheuristic algorithms [23], [24] have demonstrated adaptability in addressing AoI optimization problems. The aforementioned studies on AoI fail to fully optimize energy efficiency, communication resource allocation, and computing resource allocation—addressing only subsets of these dimensions and thus leading to inefficient utilization, higher operational costs, and subsequent degraded overall system performance. In addition, they are confined to the EC environment and did not fully leverage the potential of cloud resources, resulting in limited scalability and flexibility.

TABLE II
SUMMARY OF KEY NOTATIONS AND DEFINITIONS

Notations	Descriptions
$\mathcal{D}, \mathcal{S}, \mathcal{K}$	Set of EDs, ESs, task numbers
D_i, S_j	i-th ED, j-th ES
$\mathcal{D}'_j$	Set of EDs associated with ES S_j
E_i	Energy consumption of ED D_i
$A_i(t)$	AoI of ED D_i at time t
A_i^{ave}	Average AoI of ED D_i
$A_{\text{imp}}/E_{\text{imp}}$	Normalized AoI/energy consumption improvement
Q_{sys}	Utility function about AoI and energy consumption
$\widetilde{A}/\widetilde{E}$	Overall average AoI/energy consumption
B_j^{edge}	Communication capability of ES S_j
$T_{i,k}^{\text{rel}}/T_{i,k}^{\text{fin}}$	Time when task $\tau_{i,k}$ is generated/completed
σ	Additive white gaussian noise power
γ	Weight coefficient to show preference between AoI and energy consumption
K_i	Total number of tasks on ED D_i
g_i	Channel gain of ED D_i
$\tau_{i,k}$	k-th task of ED D_i
$\delta_{i,k}$	Data size of task $\tau_{i,k}$
$\omega_{i,k}$	Processing density of task $\tau_{i,k}$
$d_{i,k}$	Completion deadline of task $\tau_{i,k}$
$t_{i,k}$	Processing latency of task $\tau_{i,k}$
$r_{i,k,j}$	Rate of sending task $\tau_{i,k}$ to ES S_j
$r^{\text{wl}}/r^{\text{wd}}$	Wireless/wired transmission rate
p_i^{trans}	Transmission power of ED D_i
$e_{i,k}/\widetilde{e_{i,k}}/\widehat{e_{i,k}}$	Energy consumption when executing task $\tau_{i,k}$ by local/edge/cloud
$f_i/f_j^{\text{edge}}/f^{\text{cloud}}$	Computing capability of ED D_i/ES S_j/cloud (in CPU cycles)
$\widetilde{t_{i,k}^{\text{trans}}}/\widehat{t_{i,k}^{\text{trans}}}$	Communication latency when offloading task $\tau_{i,k}$ to edge/cloud
$t_{i,k}^{\text{exe}}/\widetilde{t_{i,k}^{\text{exe}}}/\widehat{t_{i,k}^{\text{exe}}}$	Computation latency of task $\tau_{i,k}$ executed by local/edge/cloud
Decisions	**Descriptions**
$B_{i,k,j}^{\text{edge}}/f_{i,k,j}^{\text{edge}}$	Communication/computing resource allocated to task $\tau_{i,k}$ by ES S_j
$(x_{i,k}^{\text{L}}, x_{i,k}^{\text{E}}, x_{i,k}^{\text{C}})$	Offloading variables indicating how to process $\tau_{i,k}$
$w_{i,k}$	Waiting time between the completion of task $\tau_{i,k-1}$ and the readiness of task $\tau_{i,k}$

Summary: In light of the above, we explore the task offloading and resource allocation problem in EECC environments, aiming to minimize both AoI and energy consumption under the deadline and resource constraints.

III. PRELIMINARY

This section first presents the EECC architecture, followed by an introduction to the preliminaries of computation latency, communication latency, energy consumption, and AoI. The relevant notations are summarized in Table II.

A. EECC Architecture

As shown in Fig. 1, our EECC system consists of a cloud, M ESs, and N EDs. Denote the set of ESs by $\mathcal{S} = \{S_j | 1 \leq j \leq M\}$ and the set of EDs by $\mathcal{D} = \{D_i | 1 \leq i \leq N\}$. For each ES S_j, the computation and communication capacities are denoted as f_j^{edge} and B_j^{edge}, respectively. Each ED D_i possesses a computing capability f_i and is associated with the nearest ES. The cloud's computing capability is considered infinite, providing each offloaded task with a consistent computing resource f^{cloud} [6]. The set of EDs associated with ES S_j is represented as $\mathcal{D}'_j$. Following [18], we assume that each ED D_i generates K_i tasks sequentially, with new task generated only

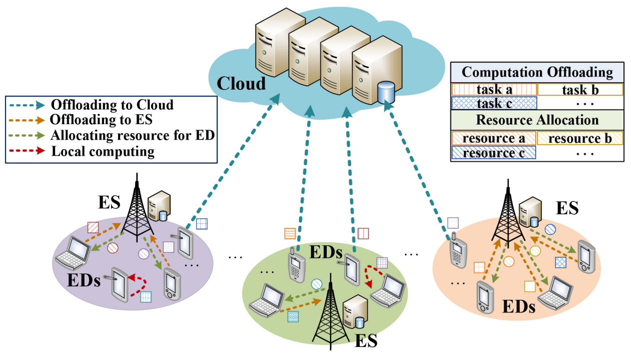


Fig. 1. EECC framework.

after the current one is completed[3]. Let $\Gamma_i = \{\tau_{i,k} | 1 \leq k \leq K_i\}$ represent the set of tasks generated by ED D_i, with $\tau_{i,k}$ being the kth task on ED D_i. Each task $\tau_{i,k}$ is characterized by a six-tuple $(\delta_{i,k}, \omega_{i,k}, d_{i,k}, w_{i,k}, T_{i,k}^{\text{rel}}, T_{i,k}^{\text{fin}})$, where $\delta_{i,k}$ is the data size, $\omega_{i,k}$ is the processing density, $d_{i,k}$ is the completion deadline, $w_{i,k}$ is the waiting time between the completion of $\tau_{i,k-1}$ and the generation of $\tau_{i,k}$, and $T_{i,k}^{\text{rel}}$ and $T_{i,k}^{\text{fin}}$ are the generation and completion times, respectively.

Tasks can be processed in three ways: locally on the ED, offloaded to an ES, or offloaded to the cloud. The offloading strategy for task $\tau_{i,k}$ is denoted by the tuple $(x_{i,k}^L, x_{i,k}^E, x_{i,k}^C)$, where $x_{i,k}^L, x_{i,k}^E, x_{i,k}^C \in \{0, 1\}$. Specifically, $x_{i,k}^L = 1$ indicates local execution at ED D_i with a local computing resource f_i, $x_{i,k}^E = 1$ indicates offloading to ES S_j with computation and communication resources f_j^{edge} and B_j^{edge}, and $x_{i,k}^C = 1$ denotes offloading to the cloud with a computing resource f^{cloud} allocated. EDs communicate with the cloud via both wireless and wired connections, with transmission rates denoted as r^{wl} for wireless and r^{wd} for wired [6].

B. Communication Latency

As mentioned above, tasks can be offloaded either to ESs or to the cloud. Therefore, we categorize the communication latency into two types: 1) end-to-edge communication latency and 2) end-to-cloud communication latency.

End-to-Edge Communication Latency: For the EDs associated with ES S_j, i.e., $D_i \in \mathcal{D}'_j$, they can simultaneously offload their tasks to the server. We use orthogonal frequency division multiple access (OFDMA) technology for end-to-edge communication. When ED D_i offloads its task $\tau_{i,k}$ to the corresponding ES E_j, the bandwidth $B_{i,k,j}^{\text{edge}}$ is allocated. According to Shannon's theorem [34], [35], the transmission rate of ED D_i when sending task $\tau_{i,k}$ to ES E_j is computed as

$$r_{i,k,j} = B_{i,k,j}^{\text{edge}} \log_2 \left(1 + \frac{p_i^{\text{trans}} g_i}{\sigma + \sum_{D_{\hat{i}} \in \mathcal{D}'_j, x_{\hat{i},k}^E = 1, \hat{i} \neq i} p_{\hat{i}}^{\text{trans}} g_{\hat{i}}} \right) \quad (1)$$

where σ is the additive white gaussian noise power, p_i^{trans} is the transmission power of D_i, and g_i is the channel gain. Then,

[3]In practice, IoT, smart home, and industrial devices generate tasks sequentially—spacing them to avoid resource contention and address data dependencies or hardware constraints.

the transmission latency of offloading task $\tau_{i,k}$ with data size $\delta_{i,k}$ to ES E_j can be formulated as

$$\widetilde{t_{i,k}^{\text{trans}}} = \frac{\delta_{i,k}}{r_{i,k,j}}. \tag{2}$$

End-to-Cloud Communication Latency: When ED D_i chooses to offload its task $\tau_{i,k}$ directly to the cloud, the end-to-cloud transmission latency includes both wireless and wired transmission latencies [6], [36]. Thus, the transmission latency of offloading $\tau_{i,k}$ with data size $\delta_{i,k}$ to the cloud is derived as

$$\widehat{t_{i,k}^{\text{trans}}} = \frac{\delta_{i,k}}{r^{\text{wl}}} + \frac{\delta_{i,k}}{r^{\text{wd}}} \tag{3}$$

where r^{wl} and r^{wd} are the wireless transmission rate and the wired transmission rate, respectively.

C. Computation Latency

In our system, tasks can be executed locally, offloaded to ESs, or offloaded to the cloud. Accordingly, the computation model is categorized into three types: local, edge, and cloud computation, as follows.

Local Computation: When ED D_i executes its task $\tau_{i,k}$ locally, the computation latency is calculated as

$$t_{i,k}^{\text{exe}} = \frac{\delta_{i,k}\omega_{i,k}}{f_i}. \tag{4}$$

Edge Computation: When ED D_i offloads its task $\tau_{i,k}$ to the corresponding ES E_j, the allocated computational resource is $f_{i,k,j}^{\text{edge}}$. The computation latency is formulated as

$$\widetilde{t_{i,k}^{\text{exe}}} = \frac{\delta_{i,k}\omega_{i,k}}{f_{i,k,j}^{\text{edge}}}. \tag{5}$$

Cloud Computation: The cloud has sufficient computing resources to compute multiple tasks in parallel [6]. When ED D_i chooses to offload its task $\tau_{i,k}$ to the cloud, the computation latency is derived as

$$\widehat{t_{i,k}^{\text{exe}}} = \frac{\delta_{i,k}\omega_{i,k}}{f^{\text{cloud}}}. \tag{6}$$

Considering both communication and computation processes, the processing latency of task $\tau_{i,k}$ is defined as

$$t_{i,k} = x_{i,k}^{L} t_{i,k}^{\text{exe}} + x_{i,k}^{E}\left(\widetilde{t_{i,k}^{\text{trans}}} + \widetilde{t_{i,k}^{\text{exe}}}\right) + x_{i,k}^{C}\left(\widehat{t_{i,k}^{\text{trans}}} + \widehat{t_{i,k}^{\text{exe}}}\right). \tag{7}$$

In our task generation model, each task $\tau_{i,k}$ is generated after $\tau_{i,k-1}$ completes and a variable waiting time $w_{i,k}$ elapses [18]. Therefore, the generation time $T_{i,k}^{\text{rel}}$ (when the task becomes ready for processing) corresponds to the earliest start time[4], and the completion time $T_{i,k}^{\text{fin}}$ of task $\tau_{i,k}$ is the sum of its generation time and its processing latency, which is

$$T_{i,k}^{\text{fin}} = T_{i,k}^{\text{rel}} + t_{i,k}. \tag{8}$$

[4]In our model, each task $\tau_{i,k}$ is represented by a tuple, including its generation time $T_{i,k}^{\text{rel}}$ and completion time $T_{i,k}^{\text{fin}}$. $T_{i,k}^{\text{rel}}$ is determined sequentially as the completion time of task $\tau_{i,k-1}$ plus a controllable intertask waiting time $w_{i,k}$. As such, it serves as the earliest possible start time for task $\tau_{i,k}$.

D. Energy Consumption

The energy consumption of each ED can be computed separately in three scenarios based on the offloading strategy.

1) *Energy Consumption for Local Computation:* When ED D_i executes its task $\tau_{i,k}$ locally, its energy consumption only includes computational energy consumption. This energy depends on the total number of CPU cycles required and the local CPU frequency f_i. Specifically, the energy consumption can be expressed as

$$e_{i,k} = \kappa {f_i}^2 C = \kappa {f_i}^2 \delta_{i,k}\omega_{i,k} = \kappa {f_i}^3 t_{i,k}^{\text{exe}} \tag{9}$$

where κ is a frequency-related coefficient, $\kappa {f_i}^2$ is energy consumption per CPU cycle [7], $C = \delta_{i,k}\omega_{i,k}$ denotes the total CPU cycles for task $\tau_{i,k}$ with $\delta_{i,k}$ representing the data size and $\omega_{i,k}$ denoting the processing density, and $t_{i,k}^{\text{exe}} = (\delta_{i,k}\omega_{i,k}/f_i)$ is substituted from (4).[5]

2) *Energy Consumption for Offloading to ES:* When ED D_i offloads its task $\tau_{i,k}$ to the corresponding ES, its energy consumption only includes transmission energy, which is

$$\widetilde{e_{i,k}} = \widetilde{t_{i,k}^{\text{trans}}} p_i^{\text{trans}} \tag{9}$$

where p_i^{trans} is the transmission power of ED D_i.

3) *Energy Consumption for Offloading to Cloud:* When ED D_i offloads its task $\tau_{i,k}$ to the cloud, its energy consumption also only includes transmission energy consumption, which is

$$\widehat{e_{i,k}} = \widehat{t_{i,k}^{\text{trans}}} p_i^{\text{trans}}. \tag{10}$$

As discussed above, the energy consumption of executing all tasks on ED D_i is expressed as

$$E_i = \sum_{k=1}^{K_i}\left(x_{i,k}^{L} e_{i,k} + x_{i,k}^{E}\widetilde{e_{i,k}} + x_{i,k}^{C}\widehat{e_{i,k}}\right). \tag{11}$$

Subsequently, the total energy consumption of all EDs can be derived as

$$\widetilde{E} = \sum_{D_i \in \mathcal{D}} E_i. \tag{12}$$

E. Age of Information

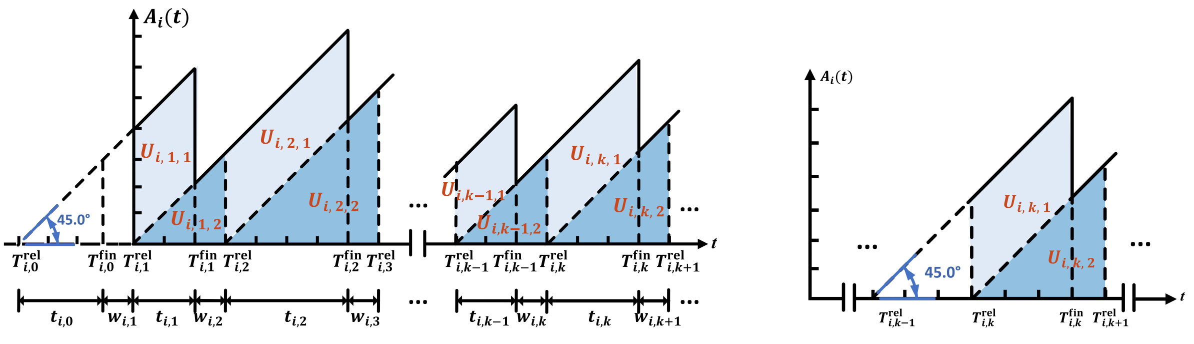
The AoI of ED D_i corresponds to the time elapsed by the latest completed task generated on that device. Therefore, the AoI of ED D_i at time t can be expressed as

$$\mathrm{A}_i(t) = t - T_{i,\max}^{\text{rel}}(t) \tag{13}$$

where $T_{i,\max}^{\text{rel}}(t)$ denotes the generation time of the latest completed task on ED D_i.

Fig. 2(a) gives an example to show the evolution of AoI of ED D_i. As shown in the figure, AoI increases linearly with time until any task is completed. The linear relationship

[5]This formulation follows the commonly used assumption in DVFS-based processor modeling [7], where the energy consumed per CPU cycle is proportional to ${f_i}^2$. Given that the number of CPU cycles required for the task $\tau_{i,k}$ is $\delta_{i,k}\omega_{i,k}$, the total local energy consumption is initially expressed as $e_{i,k} = \kappa {f_i}^2 \delta_{i,k}\omega_{i,k}$. Using (4), $e_{i,k} = \kappa {f_i}^3 t_{i,k}^{\text{exe}}$ is obtained, which reveals the cubic relationship between energy and frequency due to the inverse dependency of latency and frequency.


$A_i(t)$
45.0°
$U_{i,1,1}$
$U_{i,1,2}$
$U_{i,2,1}$
$U_{i,2,2}$
$U_{i,k-1,1}$
$U_{i,k-1,2}$
$U_{i,k,1}$
$U_{i,k,2}$
$T^{\text{rel}}_{i,0}$ $T^{\text{fin}}_{i,0}$ $T^{\text{rel}}_{i,1}$ $T^{\text{fin}}_{i,1}$ $T^{\text{rel}}_{i,2}$ $T^{\text{fin}}_{i,2}$ $T^{\text{rel}}_{i,3}$
$T^{\text{rel}}_{i,k-1}$ $T^{\text{fin}}_{i,k-1}$ $T^{\text{rel}}_{i,k}$ $T^{\text{fin}}_{i,k}$ $T^{\text{rel}}_{i,k+1}$
$t_{i,0}$ $w_{i,1}$ $t_{i,1}$ $w_{i,2}$ $t_{i,2}$ $w_{i,3}$
$t_{i,k-1}$ $w_{i,k}$ $t_{i,k}$ $w_{i,k+1}$
t

(a) (b)

Fig. 2. Example to show the evolution of AoI of ED D_i. (a) Evolution of AoI of ED D_i. (b) AoI of task $\tau_{i,k}$ on ED D_i.

between AoI and time results in an isosceles right triangle formed by the area bounded by the function and the x-axis. This occurs because the function's linear variation creates equal slopes on both sides of the vertex. For example, starting from time $t = T^{\text{rel}}_{i,1}$, AoI increases linearly until task $\tau_{i,1}$ is completed, and AoI drops to $T^{\text{fin}}_{i,1} - T^{\text{rel}}_{i,1}$. Obviously, the function of AoI is continuous. The average AoI of ED D_i is equal to the area of AoI function divided by the latest task completion time, which can be expressed as

$$A^{\text{ave}}_i = \frac{1}{T^{\text{fin}}_{i,K_i}} \int_0^{T^{\text{fin}}_{i,K_i}} \mathrm{A}_i(t)dt. \tag{14}$$

IV. Problem Definition

In this section, we calculate the average AoI of the EECC system and formulate the optimization problem.

A. Average AoI Calculation

Based on the definition given in (14), we deduce the average AoI of the EECC system as follows, which is calculated as the sum of the average AoI of each ED. As discussed in [2], the average AoI of each ED depends not only on the processing latency of tasks given in (7) but also on the waiting time between when the previous task finished and the next task was generated. The waiting time of ED D_i between task $\tau_{i,k-1}$ and $\tau_{i,k}$ can be expressed as $w_{i,k} = T^{\text{rel}}_{i,k} - T^{\text{fin}}_{i,k-1}$.[6] For ED D_i, to calculate the average AoI, the total AoI of each ED must first be computed and then divided by the time point at which all K_i tasks are completed. As depicted in Fig. 2(a), the total AoI of D_i corresponds to the area enclosed by the sawtooth function and the x-axis. Using each task's generation time $T^{\text{rel}}_{i,k}$ as the dividing point, this total area can be partitioned into multiple smaller regions, each representing the AoI of a task. For task $\tau_{i,k}$, its AoI comprises the area $U_{i,k,1}$ of a parallelogram (light blue) and the area $U_{i,k,2}$ of an isosceles right triangle (dark blue) as exemplified in Fig. 2(b). Based on the above, we have

$$\begin{aligned}
A^{\text{ave}}_i &= \frac{1}{T^{\text{fin}}_{i,K_i}} \int_0^{T^{\text{fin}}_{i,K_i}} \mathrm{A}_i(t)dt = \frac{1}{T^{\text{fin}}_{i,K_i}} \sum_{k=1}^{K_i} \left(U_{i,k,1} + U_{i,k,2}\right) \\
&= \frac{1}{T^{\text{fin}}_{i,K_i}} \left(\sum_{k=1}^{K_i} \left(T^{\text{rel}}_{i,k} - T^{\text{rel}}_{i,k-1}\right)\left(T^{\text{fin}}_{i,k} - T^{\text{rel}}_{i,k}\right) \right. \\
&\quad \left. + \sum_{k=1}^{K_i-1} \frac{\left(T^{\text{rel}}_{i,k+1} - T^{\text{rel}}_{i,k}\right)^2}{2} + \frac{\left(T^{\text{fin}}_{i,K_i} - T^{\text{rel}}_{i,K_i}\right)^2}{2} \right) \\
&= \frac{1}{\sum_{k=1}^{K_i}\left(t_{i,k} + w_{i,k}\right)} \left(\sum_{k=1}^{K_i} \left(t_{i,k-1} + w_{i,k}\right) t_{i,k} \right. \\
&\quad \left. + \sum_{k=1}^{K_i-1} \frac{\left(t_{i,k} + w_{i,k+1}\right)^2}{2} + \frac{{t_{i,K_i}}^2}{2} \right).
\end{aligned} \tag{15}$$

Thus, the average AoI of all EDs is expressed as

$$\widetilde{A} = \sum_{D_i \in \mathcal{D}} A^{\text{ave}}_i. \tag{16}$$

B. Problem Formulation

In this article, we aim to minimize the system average AoI and total energy consumption of all EDs under constraints of computing resources, communication resources, and task deadlines. However, since AoI and energy consumption are measured in different dimensions, we normalize both metrics. Following the approach proposed in [37], we define the normalized AoI and energy efficiency improvement for a given device set $\mathcal{D}$ and task number set $\mathcal{K}$ as

$$A_{\text{imp}}(\mathcal{D}, \mathcal{K}) = \frac{A_{\text{loc}}(\mathcal{D}, \mathcal{K}) - \widetilde{A}}{A_{\text{loc}}(\mathcal{D}, \mathcal{K})} \tag{17}$$

$$E_{\text{imp}}(\mathcal{D}, \mathcal{K}) = \frac{E_{\text{loc}}(\mathcal{D}, \mathcal{K}) - \widetilde{E}}{E_{\text{loc}}(\mathcal{D}, \mathcal{K})} \tag{18}$$

where $\mathcal{K} = \{K_i | D_i \in \mathcal{D}\}$, $A_{\text{loc}}(\mathcal{D}, \mathcal{K})$ and $E_{\text{loc}}(\mathcal{D}, \mathcal{K})$ denote the AoI and energy consumption of executing all tasks locally, respectively. According to (9) and (15), the AoI and energy consumption for locally executing all tasks are expressed as

$$\begin{aligned}
A_{\text{loc}}(\mathcal{D}, \mathcal{K}) &= \sum_{D_i \in \mathcal{D}} \frac{1}{\sum_{k=1}^{K_i}\left(t^{\text{exe}}_{i,k} + w_{i,k}\right)} \\
&\quad \left(\frac{{t^{\text{exe}}_{i,K_i}}^2}{2} + \sum_{k=1}^{K_i} \left(t^{\text{exe}}_{i,k-1} + w_{i,k}\right) t^{\text{exe}}_{i,k} \right. \\
&\quad \left. + \sum_{k=1}^{K_i-1} \frac{\left(t^{\text{exe}}_{i,k} + w_{i,k+1}\right)^2}{2} \right)
\end{aligned} \tag{19}$$

$$E_{\text{loc}}(\mathcal{D}, \mathcal{K}) = \sum_{D_i \in \mathcal{D}} \sum_{k=1}^{K_i} \kappa f_i^3 t^{\text{exe}}_{i,k}. \tag{20}$$

[6]For consistency, we define $w_{i,1}$ but set it to 0 since it does not affect the AoI calculations from time $t = 0$ to T^{fin}_{i,K_i}.

Based on the prior discussion, we define the utility function Q_{sys} to measure the reduced AoI and energy consumption as

$$Q_{\text{sys}} = \gamma A_{\text{imp}}(\mathcal{D}, \mathcal{K}) + (1-\gamma) E_{\text{imp}}(\mathcal{D}, \mathcal{K}) \tag{21}$$

where γ is a constant within the range of $(0, 1)$ and can be used to control the preference for the average AoI and energy consumption of our system. Clearly, the larger Q_{sys}, the more energy consumption and AoI are reduced. Naturally, minimizing the average AoI and energy consumption can be achieved by maximizing the utility Q_{sys}.

The utility Q_{sys} maximization problem is formalized as

$$\max_{\mathbf{x},\mathbf{w},\mathbf{B}^{\text{edge}},\mathbf{F}^{\text{edge}}} : \quad Q_{\text{sys}} \tag{22}$$

$$\text{s.t.: } C1\text{: } x_{i,k}^{L} + x_{i,k}^{E} + x_{i,k}^{C} = 1$$
$$\forall i \in \{1, 2, \ldots, N\} \quad \forall k \in \{1, 2, \ldots, K_i\} \tag{23}$$
$$C2\text{: } \sum_{D_i \in \mathcal{D}'_j} x_{i,k}^{E} B_{i,k,j}^{\text{edge}} \le B_j^{\text{edge}}$$
$$\forall k \in \{1, 2, \ldots, K_i\} \quad \forall j \in \{1, 2, \ldots, M\} \tag{24}$$
$$C3\text{: } \sum_{D_i \in \mathcal{D}'_j} x_{i,k}^{E} f_{i,k,j}^{\text{edge}} \le f_j^{\text{edge}}$$
$$\forall k \in \{1, 2, \ldots, K_i\} \quad \forall j \in \{1, 2, \ldots, M\} \tag{25}$$
$$C4\text{: } t_{i,k} \le d_{i,k}$$
$$\forall i \in \{1, 2, \ldots, N\} \quad \forall k \in \{1, 2, \ldots, K_i\} \tag{26}$$
$$C5\text{: } x_{i,k}^{L}, x_{i,k}^{E}, x_{i,k}^{C} \in \{0, 1\}$$
$$\forall i \in \{1, 2, \ldots, N\} \quad \forall k \in \{1, 2, \ldots, K_i\} \tag{27}$$

where $\mathbf{x} = \{(x_{i,k}^{L}, x_{i,k}^{E}, x_{i,k}^{C})\}$, $\mathbf{w} = \{w_{i,k}\}$, $\mathbf{B}^{\text{edge}} = \{B_{i,k,j}^{\text{edge}}\}$, and $\mathbf{F}^{\text{edge}} = \{f_{i,k,j}^{\text{edge}}\}$. $C1$ ensures that any task $\tau_{i,k}$ can only be executed by one entity. $C2$ and $C3$ guarantee the communication and computing resources allocated to EDs in $\mathcal{D}'_j$ by ES S_j cannot exceed its capacity. $C4$ indicates that any task $\tau_{i,k}$ should be finished before its deadline.

Obviously, the problem formalized above is a MINLP problem. Solving problems with an exponential solution space is inherently time-consuming. While standard particle swarm optimization (PSO) is valued for its adaptability and efficiency in nonlinear optimization, it faces limitations in our high-dimensional 0–1 discrete offloading problem due to overshooting and weak local search. To mitigate this, we adopt SPSO, as proposed in [38], which slows particle velocities to stabilize trajectories and enhance fine-grained search. This makes it particularly suited for discrete decision spaces, improving the likelihood of identifying globally optimal configurations.[7] The details of our SPSO-based scheme are provided in the next section.

V. Proposed SPSO-Based Approach

This section first introduces the basic concepts of SPSO, then develops a two-phase task offloading and resource allocation scheme based on SPSO.

[7]By slowing down movement, SPSO allows particles to better probe adjacent discrete configurations, reducing the chance of skipping over viable solutions and improving robustness against premature convergence in rugged search spaces.

A. Basics of SPSO

The SPSO algorithm is an optimization technique inspired by swarm intelligence. It simulates group behaviors, such as bird foraging and fish migration, to find optimal solutions through information sharing among individuals. The basic idea of the SPSO algorithm is to continuously approach the optimal solution in the solution space by updating the velocity and position of particles. The particle swarm, denoted as Ψ, contains $|\Psi|$ particles. Each particle ψ_p $(1 \le p \le |\Psi|)$ represents a potential solution and has two properties: position ξ_p and velocity v_p. The position of a particle represents its current solution, while the velocity determines the direction and distance of the particle's movement in the next evolution.

The updating rule of particles consists of two parts: the historical best position of the particle itself, called the personal best (pBest) position (ξ_p^{pBest}), and the best position within the entire swarm, known as the global best (gBest) position (ξ^{gBest}). In each iteration, particles adjust their moving speed and update their positions based on both ξ_p^{pBest} and ξ^{gBest}. Through continuous iterations, the particle swarm gradually converges to the optimal solution. Specifically, the updating rules of the velocity vector and position vector of ψ_p in each iteration are

$$v_p = \varepsilon v_p + c_1 r_1 \text{sgn}\Big(\xi_p, \xi_p^{\text{pBest}}\Big)\lambda + c_2 r_2 \text{sgn}\Big(\xi_p, \xi^{\text{gBest}}\Big)\lambda \tag{28}$$

$$\xi_p = \xi_p + v_p \tag{29}$$

where ε is the inertia weight. c_1 and c_2 are the self-learning and social learning factors, respectively. r_1 and r_2 are used to enhance the randomness of the search process. λ is a small positive constant regulating the step size of the movement. $\text{sgn}(\cdot)$ is used to control the direction of particle movement.

Next, we customize the position vectors, velocity vectors, updating rules, parameter settings, and the fitness function according to the characteristics of the problem being studied, where the fitness function is used to measure the quality of each solution.

B. Proposed SPSO-Based Algorithm

Overview: In our system, tasks generated by EDs can be directly offloaded to either the ES or the cloud for execution under deadline constraints. Inspired by a similar approach in [6], we address our task offloading and resource allocation problem in two phases. In the first phase, we simplify the EECC architecture by focusing solely on end-edge offloading. The objective in this phase is to determine the task offloading strategy $\mathbf{x}$ which only involves whether the task is executed locally or offloaded to an ES for execution (i.e., deciding whether $x_{i,k}^{L}$ or $x_{i,k}^{E}$ takes the value of 1 or 0), task generation waiting time $\mathbf{w}$, and resource allocation $\mathbf{B}^{\text{edge}}$ and $\mathbf{F}^{\text{edge}}$ of the ESs, with the aim of maximizing the system utility Q_{sys} for end-edge offloading. In the second phase, we introduce cloud offloading and iteratively adjust the strategies obtained from the first phase to further enhance system utility. For each task, we compare the utility of offloading it to the cloud against the current optimal utility. If cloud offloading yields a higher value, the task is offloaded to the cloud. Meanwhile, as the task no longer competes for edge resources, prompting a re-evaluation of the end-edge offloading decisions for remaining tasks based on the first phase's logic. This iterative process continues until no further improvement can be achieved, meaning that the offloading strategies stabilize. The final decisions for end–edge–cloud offloading and resource allocation are hence derived.

1) Phase 1: Utility Maximization for End-Edge Offloading and Resource Allocation: We analyze the system characteristics and abstract them into noninterfering decision areas, represented by $\mathbb{A} = \{\mathcal{A}_j | 1 \leq j \leq M\}$. Next, we customize position vectors, velocity vectors, update rules, parameter settings, and the fitness function in the SPSO algorithm for each $\mathcal{A}_j$. It is important to note that, in addition to traditional updating rules for continuous decision variables, our update rules also necessitate adaptations for discrete decision variables $\mathbf{x}$ to suit our studied problem.

Decision Area: According to the system model in Section III, there is no resource competition between the EDs associated with different ESs. Thus, we can optimize the task offloading and resource allocation strategy of EDs associated with a certain ES S_j and traverse M ESs to obtain the final strategy for the entire system. For clarity, we define the following notion: $\mathcal{A}_j$ is the smallest decision unit of EECC system, including the ES S_j, the set of EDs $\mathcal{D}'_j$ associated with S_j, and the corresponding set of generated tasks $\Gamma'_j = \{\tau_{i,\zeta} | \forall D_i \in \mathcal{D}'_j, 1 \leq \zeta \leq K_i\}$. Suppose that the current iteration number is k, $\mathcal{D}'_{j,k} = \{D_i | D_i \in \mathcal{D}'_j, K_i \geq k\}$ is the set of EDs which have tasks to execute, and the number of EDs in $\mathcal{D}'_{j,k}$ is $N'_{j,k} = |\mathcal{D}'_{j,k}|$. The set of pending tasks of EDs in $\mathcal{D}'_{j,k}$ corresponds to $\Gamma'_{j,k} = \{\tau_{i,k} | \forall D_i \in \mathcal{D}'_{j,k}\}$, and the ith ($1 \leq i \leq N'_{j,k}$) task in $\Gamma'_{j,k}$ is denoted as τ'_i.

Setting Position and Velocity Vectors: To address the AoI and energy optimization in $\mathcal{A}_j$, the task offloading strategy $\mathbf{x}$, task generation waiting time $\mathbf{w}$ of tasks in set $\Gamma'_{j,k}$, and resources allocation $\mathbf{B}^{\text{edge}}$ and $\mathbf{F}^{\text{edge}}$ for the ES S_j must be determined. Thus, the dimensions of the position vectors and velocity vectors of particles in the swarm Ψ are both $4N'_{j,k}$. The position vector ξ_p ($1 \leq p \leq |\Psi|$) of particle ψ_p is defined as $\xi_p = (\boldsymbol{x}_p, \boldsymbol{\varpi}_p, \boldsymbol{B}_p, \boldsymbol{f}_p)$. Here, $\boldsymbol{x}_p = (x_{p,1}, \ldots, x_{p,N'_{j,k}})$ is a binary vector denoting the end-edge offloading strategy for tasks in the set $\Gamma'_{j,k}$, where $x_{p,i} = 0$ indicates τ'_i ($1 \leq i \leq N'_{j,k}$) is executed locally and $x_{p,i} = 1$ indicates τ'_i is offloaded to the ES. $\boldsymbol{\varpi}_p = (\varpi_{p,1}, \ldots, \varpi_{p,N'_{j,k}})$ denotes the waiting time for these tasks, varying in the range of $[0, \varpi_{\max}]$. $\boldsymbol{B}_p = (B_{p,1}, \ldots, B_{p,N'_{j,k}})$ and $\boldsymbol{f}_p = (f_{p,1}, \ldots, f_{p,N'_{j,k}})$ represent the communication and computing resource allocation strategy of the ES S_j, respectively. Specifically, $B_{p,i}$ is the bandwidth provided by S_j for transmitting τ'_i, ranging from $[0, B_j^{\text{edge}}]$, and $f_{p,i}$ is the computing frequency allocated by S_j for executing τ'_i, varying from $[0, f_j^{\text{edge}}]$. The corresponding velocity vector is defined as $v_p = (\boldsymbol{v}_p^{\boldsymbol{x}}, \boldsymbol{v}_p^{\boldsymbol{\varpi}}, \boldsymbol{v}_p^{\text{B}}, \boldsymbol{v}_p^{\text{f}})$. Similarly, $\boldsymbol{v}_p^{\boldsymbol{x}} = (v_{p,1}^x, \ldots, v_{p,N'_{j,k}}^x)$, $\boldsymbol{v}_p^{\boldsymbol{\varpi}} = (v_{p,1}^{\varpi}, \ldots, v_{p,N'_{j,k}}^{\varpi})$, $\boldsymbol{v}_p^{\text{B}} = (v_{p,1}^{\text{B}}, \ldots, v_{p,N'_{j,k}}^{\text{B}})$, and $\boldsymbol{v}_p^{\text{f}} = (v_{p,1}^{\text{f}}, \ldots, v_{p,N'_{j,k}}^{\text{f}})$, where $v_{p,i}^x$, $v_{p,i}^{\varpi}$, $v_{p,i}^{\text{B}}$, and $v_{p,i}^{\text{f}}$, ($1 \leq i \leq N'_{j,k}$) represent the updating speed of $x_{p,i}$, $\varpi_{p,i}$, $B_{p,i}$, and $f_{p,i}$, respectively.

Setting Updating Rules: The velocity vector v_p is updated according to (28). Due to the binary nature of $\boldsymbol{x}_p$ in ξ_p, $x_{p,i}$ ($1 \leq i \leq N'_{j,k}$) cannot be updated simply using (29). Instead, we use the sigmoid function to map the velocity variable $v_{p,i}^x$ to the interval [0, 1] [39]. The sigmoid function is defined as

$$s\left(v_{p,i}^x\right) = \frac{1}{1 + \exp\left(-v_{p,i}^x\right)}. \tag{30}$$

The mapping result $s(v_{p,i}^x)$ indicates the probability of $v_{p,i}^x = 1$, i.e., the probability of offloading task τ'_i to the ES for execution. The end-edge offloading indicator $x_{p,i}$ for τ'_i is determined by comparing $s(v_{p,i}^x)$ with a randomly generated number ϵ

$$x_{p,i} = \begin{cases} 1, \text{ if } & \epsilon \leq s\left(v_{p,i}^x\right) \\ 0, \text{ otherwise.} \end{cases} \tag{31}$$

The elements of $\boldsymbol{\varpi}_p, \boldsymbol{B}_p$, and $\boldsymbol{f}_p$ in ξ_p are continuous variables that can be updated normally. The updated equations for $\varpi_{p,i}$, $B_{p,i}$ and $f_{p,i}$ are as follows:

$$\varpi_{p,i} = \varpi_{p,i} + v_{p,i}^{\varpi} \left(1 \leq i \leq N'_{j,k}\right) \tag{32}$$

$$B_{p,i} = B_{p,i} + v_{p,i}^{\text{B}} \left(1 \leq i \leq N'_{j,k}\right) \tag{33}$$

$$f_{p,i} = f_{p,i} + v_{p,i}^{\text{f}} \left(1 \leq i \leq N'_{j,k}\right). \tag{34}$$

Setting Fitness Function: The greedy approach is used to determine the strategy for $\mathcal{A}_j$. The objective of task offloading and resource allocation for tasks in $\Gamma'_{j,k}$ is to minimize the current AoI and energy consumption. Thus, for the task set $\Gamma'_{j,k}$, the fitness function is adapted from (21) and can be expressed as

$$Q_{j,k}^{\text{fit}} = \gamma A_{\text{imp}}\left(\mathcal{D}'_{j,k}, \mathcal{K}'_{j,k}\right) + (1 - \gamma) E_{\text{imp}}\left(\mathcal{D}'_{j,k}, \mathcal{K}'_{j,k}\right) \tag{35}$$

where $\mathcal{K}'_{j,k} = \{K'_i | D_i \in \mathcal{D}'_{j,k}, K'_i = \min\{k, K_i\}\}$. The gBest particle position $\xi_{j,k}^{\text{gBest}}$ obtained by the discrete SPSO in this phase is the optimal end-edge strategy of $\mathcal{A}_j$.

Setting Key Parameters: For the proposed discrete SPSO-based algorithm, the key parameters, including the inertia weight ε, self-learning factor c_1, and social learning factor c_2 in (28) need to be optimized to achieve a superior solution and accelerate the search for the optimal solution. The proposed algorithm focuses on exploration capability when ε is large, while it focuses on exploitation capability when ε is small. Hence, we adopt linearly decreasing inertia weight [41], searching for a good position with larger ε in the early stage, while fine-tuning the search with smaller ε in the later stage. More specifically, ε decreases linearly from the initial value ($\varepsilon_{\max}$) to the final value ($\varepsilon_{\min}$) by

$$\varepsilon = \varepsilon_{\min} + \frac{I - g}{I}(\varepsilon_{\max} - \varepsilon_{\min}) \tag{36}$$

where g and I represent the current iterations and maximum iterations, respectively. As for the learning factors, to balance the algorithm's exploration and exploitation performance, it is recommended that the values of c_1 and c_2 should be varied over the iterations. In the early stage, to prevent particles from quickly converging to local optima and encourage extensive exploration in the global domain, we set a relatively large value for c_1 and a smaller value for c_2. In the later stage, to facilitate rapid and accurate convergence of particles to the global optimum, we adjust c_1 to a smaller value and c_2 to a larger value. Thus, we construct c_1 as a monotonically decreasing function and c_2 as a monotonically increasing function [40]. The expressions are as follows:

$$c_1 = 2\sin^2\left(\frac{\pi}{2}\left(1 - \frac{g}{I}\right)\right) \tag{37}$$

$$c_2 = 2\sin^2\left(\frac{\pi g}{2I}\right). \tag{38}$$

The pseudo-code of our discrete SPSO-based end-edge offloading utility maximization method is given in *Algorithm 1*. Initially, the position and velocity vectors of all particles in the swarm Ψ are randomly initialized (line 1). Lines 2–13 calculate the fitness of each particle ψ_p to initialize its pBest position ξ_p^{pBest} and corresponding fitness Q_p^{pBest}. Specifically, lines 3 and 4 set the fitness $Q_{j,k}^{\text{fit}}(\psi_p)$ to -1 when violating the resource constraints. The processing time of each task in $\Gamma'_{j,k}$ is calculated in line 6. If at least one task violates the timing constraint, the fitness $Q_{j,k}^{\text{fit}}(\psi_p)$ is set to -1 (lines 7 and 8). Lines 10 and 12 calculate the fitness of particles that satisfy all constraints. The algorithm then sets the initial particle as the pBest particle in line 13. The particle with maximum fitness is set to the gBest particle (line 14). Lines 15–22 describe the evolution process of the pBest particle and the gBest particle. The velocity and position vector of each particle are updated in line 16. For each particle ψ_p, line 18 calculates its new fitness in the same manner as lines 3–12. Lines 19 and 20 update the pBest particle if a better position is found. Similarly, lines 21 and 22 check whether to update the gBest particle. After all the evolutions, the optimal end-edge task offloading and resource allocation strategy and its corresponding fitness are obtained.

2) Phase 2: Iterative Adjustment and Optimization With Cloud Offloading: In this phase, we continue to use the Decision Area ($\mathcal{A}_j$) as the smallest decision unit. After *Phase 1*, we obtain the end-edge optimal strategy $\xi_{j,k}^{\text{gBest}}$, which serves as the baseline configuration for further refinement. Although this strategy is already well optimized, it may lack adaptability to dynamic task demands and heterogeneous resource availability, and it does not fully explore the potential benefits of CC. To refine $\xi_{j,k}^{\text{gBest}}$, we introduce cloud offloading and evaluate each task individually to determine whether it is better to maintain the existing strategy or offload the task to the cloud for processing. Note that when a specific task τ'_i in $\Gamma'_{j,k}$ is offloaded to cloud, the other tasks in $\Gamma'_{j,k}$ keep the strategy in $\xi_{j,k}^{\text{gBest}}$. Assuming that the cloud has sufficient communication and computing resources, this phase involves only one decision variable: the task generation waiting time $\mathbf{w}$. Thus, in the configuration of SPSO, the search space dimension is set to 1. The position vector is $\xi_p = (\varpi_{p,1})$, where $\varpi_{p,1}$ denotes the generation waiting time of the next task. The velocity vector is $v_p = (v_{p,1}^{\varpi})$, where $v_{p,1}^{\varpi}$ controls the searching direction and speed. The fitness function remains the same as in *Phase 1*.

Algorithm 1: Utility Maximization for End-Edge Offloading and Resource Allocation

```
   Input: $\mathcal{A}_j$, $\Gamma'_{j,k}$, $B_j^{\text{edge}}$, $f_j^{\text{edge}}$, $I$, $\lambda$, $\varepsilon$, $c_1$, $c_2$;
   Output: Optimal strategy $\xi_{j,k}^{\text{gBest}}$ of $\mathcal{A}_j$ and the corresponding
           fitness $Q_{j,k}^{\text{gBest}}$;
   // Particle swarm initialization
1  Randomly generate each particle's position vector $\xi_p$ and
   velocity vector $v_p$ in the particle swarm $\Psi$;
2  for each particle $\psi_p \in \Psi$ do
       // Constraint Violation Handling
3      if $\sum_{D_i \in \mathcal{D}'_{j,k}} x_{p,i} B_{p,i} > B_j^{\text{edge}}$ or $\sum_{D_i \in \mathcal{D}'_{j,k}} x_{p,i} f_{p,i} > f_j^{\text{edge}}$
       then
4          $Q_{j,k}^{\text{fit}}(\xi_p) \leftharpoondown -1$;
           // Personal best (pBest) selection
5      else
6          Calculate the processing latency $t_{\tau'_i}$ of each task in $\Gamma'_{j,k}$
           using Eq. (7);
7          if $\exists \tau'_i \in \Gamma'_{j,k}$, $t_{\tau'_i} > d_{\tau'_i}$ then
8              $Q_{j,k}^{\text{fit}}(\xi_p) \leftharpoondown -1$;
9          else
10             Get $\widetilde{E}$ and $E_{\text{imp}}$ by Eqs. (12) and (18);
11             Get $\widetilde{A}$ and $A_{\text{imp}}$ by Eqs. (16) and (17);
12             Get $Q_{j,k}^{\text{fit}}(\xi_p)$ by Eq. (35);
13     $\xi_p^{\text{pBest}} \leftharpoondown \xi_p$, $Q_p^{\text{pBest}} \leftharpoondown Q_{j,k}^{\text{fit}}(\xi_p^{\text{pBest}})$;
   // Global best (gBest) selection
14 $\xi_{j,k}^{\text{gBest}} \leftharpoondown \arg\max_{\xi_p} Q_p^{\text{pBest}}$, $Q_{j,k}^{\text{gBest}} \leftharpoondown \max Q_p^{\text{pBest}}$;
   // Iterative pBest and gBest update
15 for $g = 1$ to $I$ do
16     Refresh the particle swarm $\Psi$ by updating $v_p$ and $\xi_p$ of
       each particle $\psi_p$ using Eqs. (28), (30)-(32), and (36)-(38);
17     for each particle $\psi_p \in \Psi$ do
18         Execute the operations in lines 3-12 to obtain the new
           fitness $Q_{j,k}^{\text{fit}}(\xi_p)$ of $\psi_p$;
19         if $Q_{j,k}^{\text{fit}}(\xi_p) > Q_p^{\text{pBest}}$ then
20             $Q_p^{\text{pBest}} \leftharpoondown Q_{j,k}^{\text{fit}}(\xi_p)$, $\xi_p^{\text{pBest}} \leftharpoondown \xi_p$;
21         if $Q_{j,k}^{\text{fit}}(\xi_p) > Q_{j,k}^{\text{gBest}}$ then
22             $Q_{j,k}^{\text{gBest}} \leftharpoondown Q_{j,k}^{\text{fit}}(\xi_p)$, $\xi_{j,k}^{\text{gBest}} \leftharpoondown \xi_p$;
23
```

The pseudo-code of the proposed SPSO-based iterative adjustment and optimization with cloud offloading method is presented in *Algorithm 2*. The algorithm begins by setting $\mathcal{A}_j^{\text{begin}}$ as the input $\mathcal{A}_j$ (line 1). Each task τ'_i in $\Gamma'_{j,k}$ is evaluated sequentially to determine its eligibility for cloud offloading under the given constraints (lines 3–5). If the deadline constraint is satisfied, we use SPSO to find the optimal cloud offloading strategy and the corresponding fitness $Q_{j,k}^{\text{gc}}$ for τ'_i (lines 7–16). We then compare $Q_{j,k}^{\text{gc}}$ with $Q_{j,k}^{\text{gBest}}$ obtained from *Algorithm 1*. If $Q_{j,k}^{\text{gc}}$ is superior, τ'_i will be offloaded to the cloud and removed from $\Gamma'_{j,k}$, with its generation waiting time set to $\xi_{j,k}^{\text{gc}}$ (lines 17–19). Consequently, as τ'_i exits the competition for the resources of S_j, the remaining tasks in $\Gamma'_{j,k}$ need to reoptimize using *Algorithm 1* (line 20). Based on the redecision results, the input of $\mathcal{A}_j$, including the set of tasks

Algorithm 2: Iterative Adjustment and Optimization With Cloud Offloading

```
Input: $k$, $\mathcal{A}_j$, $\Gamma'_{j,k}$, $f^{\text{cloud}}$, $r^{\text{wl}}$, $r^{\text{wd}}$ as well as the $\xi^{\text{gBest}}_{j,k}$ and $Q^{\text{gBest}}_{j,k}$ obtained by Algorithm 1;
Output: $\mathbf{x}$, $\mathbf{w}$, $\mathbf{B}^{\text{edge}}$, $\mathbf{F}^{\text{edge}}$ of $\mathcal{A}^{\text{begin}}_j$;
1  Initialization: $\mathcal{A}^{\text{begin}}_j \leftharpoondown \mathcal{A}_j$;
   // Iterative optimization for $\xi^{gBest}_{j,k}$
2  for $\tau'_i \in \Gamma'_{j,k}$ do
3      Calculate processing latency $t_{\tau'_i}$ when offloading $\tau'_i$ to the cloud by Eqs. (3) and (6);
       // Deadline constraint check
4      if $t_{\tau'_i} > d_{\tau'_i}$ then
5          Set $\mathbf{x}$, $\mathbf{w}$, $\mathbf{B}^{\text{edge}}$, $\mathbf{F}^{\text{edge}}$ for $\tau'_i$ as specified in $\xi^{\text{gBest}}_{j,k}$;
6      else
           // Search for cloud offloading strategy of $\tau'_i$ based on SPSO
7          Initialize $\Psi$ as Algorithm 1;
8          for $\psi_p \in \Psi$ do
9              Get $\widetilde{E}$ and $E_{\text{imp}}$ by Eqs. (12) and (18) based on $\xi^{\text{gBest}}_{j,k}$;
10             Get $\widetilde{A}$ and $A_{\text{imp}}$ by Eqs. (16) and (17);
11             Get $Q^{\text{fit}}_{j,k}(\xi_p)$ by Eq. (35);
12             $\xi^{\text{pBest}}_p \leftharpoondown \xi_p$, $Q^{\text{pBest}}_p \leftharpoondown Q^{\text{fit}}_{j,k}(\xi^{\text{pBest}}_p)$;
13         $\xi^{\text{gc}}_{j,k} \leftharpoondown \arg\max_{\xi_p} Q^{\text{pBest}}_p$, $Q^{\text{gc}}_{j,k} \leftharpoondown \max Q^{\text{pBest}}_p$;
14         for $g = 1$ to $I$ do
15             Update $v_p$ and $\xi_p$ of each $\psi_p$ by Eqs. (28), (32), and (36)-(38);
16             Update $\xi^{\text{pBest}}_p$, $Q^{\text{pBest}}_p$, $\xi^{\text{gc}}_{j,k}$, and $Q^{\text{gc}}_{j,k}$ in the same manner as lines 17-22 in Algorithm 1;
           // Adjust the strategy for $\tau'_i$ and re-optimize the strategy for the remaining tasks
17         if $Q^{\text{gc}}_{j,k} > Q^{\text{gBest}}_{j,k}$ then
18             $x^{\text{C}}_{\tau'_i} \leftharpoondown 1$, $w_{\tau'_i} \leftharpoondown \xi^{\text{gc}}_{j,k}$ to offload $\tau'_i$ to the cloud;
19             Remove $\tau'_i$ and its ED from $\Gamma'_{j,k}$ and $\mathcal{D}'_{j,k}$;
20             Call Algorithm 1 to re-decide for the remaining tasks in $\Gamma'_{j,k}$;
21             Update $\mathcal{A}_j$, $\xi^{\text{gBest}}_{j,k}$ and $Q^{\text{gBest}}_{j,k}$;
22         // Keep the strategy for $\tau'_i$
23         else
24             Set $\mathbf{x}$, $\mathbf{w}$, $\mathbf{B}^{\text{edge}}$, $\mathbf{F}^{\text{edge}}$ for $\tau'_i$ as in $\xi^{\text{gBest}}_{j,k}$;
```

$\Gamma'_{j,k}$ and the set of EDs $\mathcal{D}'_{j,k}$, along with $\xi^{\text{gBest}} j,k$ and $Q^{\text{gBest}}_{j,k}$, are updated (line 21). If $Q^{\text{gBest}}_{j,k}$ is preferred, the strategy for τ'_i will retain the existing strategy in $\xi^{\text{gBest}}_{j,k}$ (line 23). This process is repeated for all tasks until $\Gamma'_{j,k}$ is empty, at which point the optimal strategy for the initial input $\mathcal{A}_j$, i.e., $\mathcal{A}^{\text{begin}}_j$, is obtained.

3) Two-Phase Computation Offloading and Resource Allocation: The proposed two-phase computation offloading and resource allocation method is summarized in *Algorithm 3*. Given that no new tasks are generated until the current pending task has obtained its execution result, we can use the current pending task number of EDs as the outer loop and each $\mathcal{A}_j$ in our system as the inner loop to ensure that the processing strategies of all tasks across all EDs are decided. Specifically, in the inner loop, when the current iteration number is k, the algorithm first obtains the information of $\mathcal{A}_j$, including S_j, $\mathcal{D}'_{j,k}$, and $\Gamma'_{j,k}$ (line 4). Next, it calls *Algorithm 1* to determine the optimal end-edge execution strategy for $\mathcal{A}_j$ (line 5). Then, it calls *Algorithm 2* to adjust and further optimize the execution strategy for $\mathcal{A}_j$ obtained by the last step (line 6). After each iteration of the inner loop, the execution strategies for all pending tasks within the current round have been generated. Once the outer loop has iterated through all task iterations, the execution strategies for all tasks across the EECC system are fully determined. The utility value Q_{sys} of the EECC system is calculated with the final strategy $(\mathbf{x}, \mathbf{w}, \mathbf{B}^{\text{edge}}, \mathbf{F}^{\text{edge}})$ (line 7).

Algorithm 3: Two-phase Computation Offloading and Resource Allocation

```
Input: $\mathcal{D}$, $\mathcal{S}$, $\Gamma$;
Output: $\mathbf{x}$, $\mathbf{w}$, $\mathbf{B}^{\text{edge}}$, $\mathbf{F}^{\text{edge}}$, $Q_{\text{sys}}$;
1  $K \leftharpoondown \max\{K_i | D_i \in \mathcal{D}\}$;
2  for $k = 1$ to $K$ do
3      for $S_j \in \mathcal{S}$ do
4          Get $\mathcal{A}_j$'s information, including $S_j$, $\mathcal{D}'_{j,k}$, and $\Gamma'_{j,k}$;
5          Call Algorithm 1 to get the optimal end-edge offloading strategy $(\xi^{\text{gBest}}_{j,k}, Q^{\text{fit}}_{j,k})$ for $\mathcal{A}_j$;
6          Call Algorithm 2 to further adjust and refine $\xi^{\text{gBest}}_{j,k}$;
7  Use (21) to obtain the utility $Q_{\text{sys}}$ based on $\mathbf{x}$, $\mathbf{w}$, $\mathbf{B}^{\text{edge}}$, and $\mathbf{F}^{\text{edge}}$ associated with $\xi^{\text{gBest}}_{j,k}$;
```

4) Computational Complexity: Notably, since the decision areas $\mathcal{A}_j$ $(1 \leq j \leq M)$ are mutually independent, the overall optimization process allows parallel execution, which significantly reduces the practical computation time through code parallelization. To evaluate the computational complexity of the proposed algorithm (*Algorithm 3*), we consider the worst-case scenario when the EECC system contains only one ES $(M = 1)$.

Theorem 1: The total computational complexity of our proposed algorithm is bounded by $\mathcal{O}(KIN^2|\Psi|)$, where K is the maximum number of tasks per ED, N is the number of EDs, $|\Psi|$ is the particle population size, and I is the maximum number of SPSO iterations.

Proof: Assume that $M = 1$, i.e., the system contains only one ES. In this case, the outer loop of *Algorithm 3* iterates over K tasks in the worst case, resulting in a complexity of $\mathcal{O}(K)$. Within each iteration, the algorithm invokes both *Algorithms 1* and 2. For *Algorithm 1*, during initialization, there are $|\Psi|$ particles each with a dimension of $4N$, resulting in a complexity of $\mathcal{O}(N|\Psi|)$. In the iteration phase, the resource capacity and deadline constraint checks have a complexity of $\mathcal{O}(N)$, followed by the core fitness function Q^{fit} evaluation, which is defined as a weighted sum of the AoI and energy improvement. Specifically, the AoI-related terms A_{loc} and $\widetilde{A}$ are calculated using (15) and (19) for all devices, yielding a complexity of $\mathcal{O}(N)$. Similarly, the energy-related terms E_{loc} and $\widetilde{E}$ are computed per (20) and the execution models in (9)–(11), also with complexity $\mathcal{O}(N)$. Thus, the fitness evaluation per particle costs $\mathcal{O}(N)$. With a population size of $|\Psi|$ and I iterations, the process of iterations yields a complexity of $\mathcal{O}(IN|\Psi|)$. Thus, the total complexity of *Algorithm 1* is $\mathcal{O}(N|\Psi|) + \mathcal{O}(IN|\Psi|) = \mathcal{O}(IN|\Psi|)$. *Algorithm 2* employs a similar SPSO process but introduces additional overhead for task offloading decision adjustments. In the extreme case where optimization is required for all N tasks, its complexity becomes $\mathcal{O}(N) \cdot \mathcal{O}(IN|\Psi|) + \mathcal{O}(IN|\Psi|) = \mathcal{O}(IN^2|\Psi|)$. Hence, in *Algorithm 3*, the complexity for the inner loop becomes $\mathcal{O}(IN^2|\Psi|) + \mathcal{O}(IN|\Psi|) = \mathcal{O}(IN^2|\Psi|)$, and over K iterations, the total complexity of *Algorithm 3* is $\mathcal{O}(K) \cdot \mathcal{O}(IN^2|\Psi|) = \mathcal{O}(KIN^2|\Psi|)$. This result confirms that the proposed algorithm operates within polynomial time, making it computationally feasible for practical EECC scenarios. Moreover, the scalability of the algorithm is ensured in real-world scenarios since 1) N is inherently constrained by the physical coverage of a single ES; 2) K is bounded by the task capacity per ED; and 3) optimization parameters, such as I and $|\Psi|$ can be flexibly tuned to balance performance and efficiency. ■

VI. Evaluation

This section presents the evaluation settings and then shows the results on the simulated and real-platform experiments. Specifically, our evaluation focuses on four aspects: 1) optimization of the utility function; 2) improvement of task success rates; 3) reduction of total task completion latency; and 4) performance on real-world platforms.

A. Settings

Evaluation Environment: All simulations are performed on MATLAB R2021a under Windows 10, using a computer equipped with an 8-core Intel Core i7-7700 CPU @ 3.60 GHz and 16 GB-RAM.

Metrics: First, we select the utility function Q_{sys} (with its value normalized to the interval [0, 1]) calculated by (21) as one of the evaluation metrics. Next, we examine the task success rate in detail, which is defined as the ratio of tasks completed within the deadline to the total task number of EECC system (quantified as a percentage metric ranging from 0% to 100%). To evaluate the efficacy of the AoI-oriented optimization strategy, we else compare its makespan performance (defined as the total task completion latency $\sum_{i=1}^{N} T_{i,K}^{\text{fin}}$, where $T_{i,K}^{\text{fin}}$ is the completion time of the last task on ED D_i) against latency-oriented optimization approach.

Parameter Settings: The number of ESs, denoted as M, is set to 2, 4, and 6, while the number of EDs, denoted as N, ranges from 10 to 50 in increments of 10 (equivalent to scaling the task traffic intensity). The transmission power P_i^{trans} of ED D_i ranges from 0.1 to 0.5 J/s, and the gaussian noise power σ is set at -100 dBm [42]. The wired and wireless transmission rate, r^{wd} and r^{wl}, are set to 1.52 Mb/s. The local computing frequency f_i of ED D_i ranges from 0.5 to 1 GHz, while the cloud allocates $f^{\text{cloud}} = 10$ GHz of computing resources. The computing capacity f_j^{egde} of ES S_j ranges between 35 and 45 GHz, with a communication resource B_j^{egde} of $9.97R$ Mb/s, where $R \in [5, 10]$ [6], [44]. For each task $\tau_{i,k}$, the data size $\delta_{i,k}$ is randomly selected from 1 to 5 MB, and the processing density $\omega_{i,k}$ is randomly distributed between 1000 and 3000 cycles/bit [45]. The value of $\varepsilon_{\max}$ and $\varepsilon_{\min}$ are set as 0.9 and 0.4, respectively [41]. γ is set to 0.2, 0.5, 0.8, and 1. The relevant parameters are summarized in Table III [42], [43], [44].

TABLE III
Simulation Parameters

Parameter	Value	Parameter	Value
B_j^{edge}	$9.97 \times [5, 10]$ Mbps	$\delta_{i,k}$	[1,5] MB
M	{2,4,6}	$\omega_{i,k}$	[1000,3000] cycles/bit
N	{10,20,30,40,50}	ε_{min}	0.4
I	1800	ε_{max}	0.9
f_j^{edge}	[35,45] GHz	ϖ_{max}	800 ms
f^{cloud}	10 GHz	f_i	[0.5,1] GHz
p_i^{trans}	[0.1,0.5] J/s	σ	-100 dBm
r^{wl}	1.52 Mbps	κ	5×10^{-27}
r^{wd}	1.52 Mbps	λ	0.05

Comparative Algorithms: Existing AoI optimization schemes primarily focus on task offloading without coordinated resource allocation, which might degrade performance. To highlight this limitation, we compare our method with random full-offloading algorithm (RFOA), gray wolf optimizer (GWO), and firefly algorithm (FA), which solely optimize AoI. Moreover, to reveal the drawbacks of latency-oriented approaches that overlook AoI, we include latency-oriented computation offloading algorithm (LOCOA), which prioritizes delay but neglects information freshness. Above all, we compare the proposed method with the following algorithms.

1) RFOA employs a stochastic policy for task offloading and resource allocation, with no systematic optimization. It randomly selects task destinations (ES/cloud), communication/computing resources, and waiting times within predefined limits. This naive approach serves as a critical baseline to quantify the benefits of optimization.
2) GWO [23] is based on the gray wolf optimization algorithm and has been modified to suit the problem of our research. The algorithm iteratively updates candidate solutions to approach the optimal decision variables. Its balance between exploration and exploitation makes it a strong benchmark for our studied problem.
3) FA [24] is based on the FA and has been modified to suit the problem of our research. It employs an attraction mechanism where solutions move toward brighter peers, mimicking natural swarm behavior. Its brightness-based attraction is inherently suited to discrete-continuous hybrid search spaces.
4) LOCOA [6] employs weight-based resource allocation and derives the final solution by minimizing the weighted sum of latency and energy consumption, which is compared to demonstrate the superior performance of AoI-oriented optimization.

Real Testbed Platform: We build the EECC architecture on a real testbed as shown in Fig. 3. The specific platform settings

TABLE IV
REAL TESTBED PLATFORM SETTINGS

Role	Device	Component	CPU Frequency	Number
End	Jetson Nano	4-core ARM Cortex-A57 MPCore processor	102MHz - 1.5GHz	2
	Jetson Orin Nano	6-core Arm Cortex-A78AE v8.2 64-bit CPU	729MHz - 1.5GHz	4
	Jetson Orin NX	6-core Arm Cortex-A78AE v8.2 64-bit CPU	729MHz - 2.0GHz	3
	Jetson Xavier NX	6-core NVIDIA Carmel ARMv8.2 64-bit CPU	1.2GHz - 1.5GHz	1
Edge	Jetson AGX Orin	12-core Arm Cortex-A78AE v8.2 64-bit CPU	729MHz - 2.2GHz	1
Cloud	Workstation	Intel(R) Xeon(R) Silver 4210R	1.0GHz - 3.2GHz	1

TABLE V
LIST OF TASKS

Task Name	Profile	Task Name	Profile
ANT	Alexnet network training	IC_C10	Image classification based on CIFAR-10
EYE_SEG	Semantic segmentation of ocular images	PD_HOG	Pedestrian detection based on HOG features
FED	Facial expression detection	SA_MR	Sentiment analysis of movie reviews
FRMVF	Facial recognition and masking in video frames	SAT_HRD	Sentiment analysis of chinese text in hotel reviews
HDR_MNIST	Handwritten digit recognition based on the MNIST	TPIDS	Tongue papillae image detection and segmentation

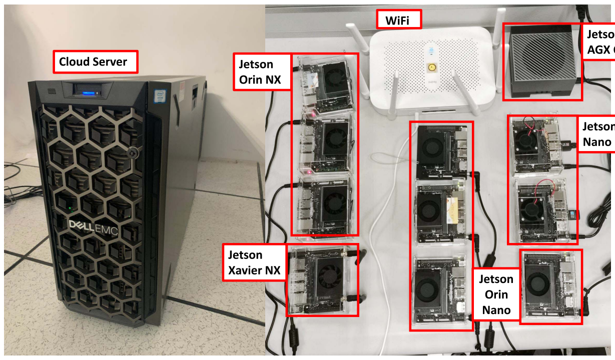


Fig. 3. Snapshot of the implemented testbed.

are elaborated in Table IV. Considering the heterogeneity at the user layer, we select four types of development boards with varying computational capabilities, i.e., Nvidia's Jetson Xavier NX, Jetson Orin NX, Jetson Nano, and Jetson Orin Nano, as EDs.

Task Settings: We designate each ED to perform five tasks, and for each ED, the tasks it performs are randomly selected from the whole task set. The task set includes a variety of classic and practical applications [46], [47], [48], [49], [50]. Detailed specifications of the available tasks are outlined in Table V.

B. Simulation Results

1) *Utility:* We first examine the utility values achieved by the proposed method and comparative algorithms (i.e., RFOA, GWO, FA) with different number of EDs for varying numbers of ESs and weight coefficients. In this evaluation, the deadline for task execution ranges from 80% to 110% of the local execution time. The comparison results are presented in Fig. 4, demonstrating that the proposed method consistently outperforms the other three, regardless of system scales and parameter configurations. As depicted in Fig. 4(a), (d), and (g), when the weight coefficient γ is set to 0.8, indicating the system prioritizes AoI, the proposed method significantly enhances utility values compared to the others. It improves utility by an average of 52.44%, reaching a peak improvement of 70.33% over RFOA, 14.25% on average and up to 15.89% over GWO, and 21.40% on average and up to 23.64% over FA. As illustrated in Fig. 4(b), (e), and (h), when γ is set to 0.5, indicating the system equally prioritizes AoI and energy consumption, the proposed method continues to outperform the others, with an average utility improvement of 34.03% (up to 46.48%) over RFOA, 13.37% (up to 15.48%) over GWO, and 16.72% (up to 19.73%) over FA. Finally, when γ is set to 0.2, indicating the system prioritizes energy consumption, the proposed method still delivers superior results. It achieves a peak enhancement of 28.38% over RFOA, 10.31% over GWO, and 14.29% over FA, as shown in Fig. 4(c), (f), and (j).

In addition, when the number of ESs is fixed, the utility of each algorithm shows a declining trend as the number of EDs increases, and this trend is progressively more pronounced with the decrease of ESs. Since the resources owned by each ES are limited, for each decision area, the competition for resources intensifies as the number of EDs increases, and the bandwidth and computing resources that can be allocated to each task decrease accordingly. This decrease affects both the transmission and execution time when tasks are offloaded to the ES, which in turn affects the average AoI and energy consumption of the EDs. Notably, when $M = 2$ and $N = 50$, the advantage of the proposed method is less pronounced compared to other cases, due to the extreme resource constraints faced by the system.

In summary, extensive experiments show that regardless of system scales and weight coefficients, the proposed method consistently outperforms the other three. It exhibits a significant advantage in terms of utility value. This is because our proposed algorithm better balances exploration and exploitation, enabling it to identify superior decisions in various scenarios within the solution space.

2) *Task Success Rates:* We then investigate the task execution success rates of the optimal solutions from four algorithms

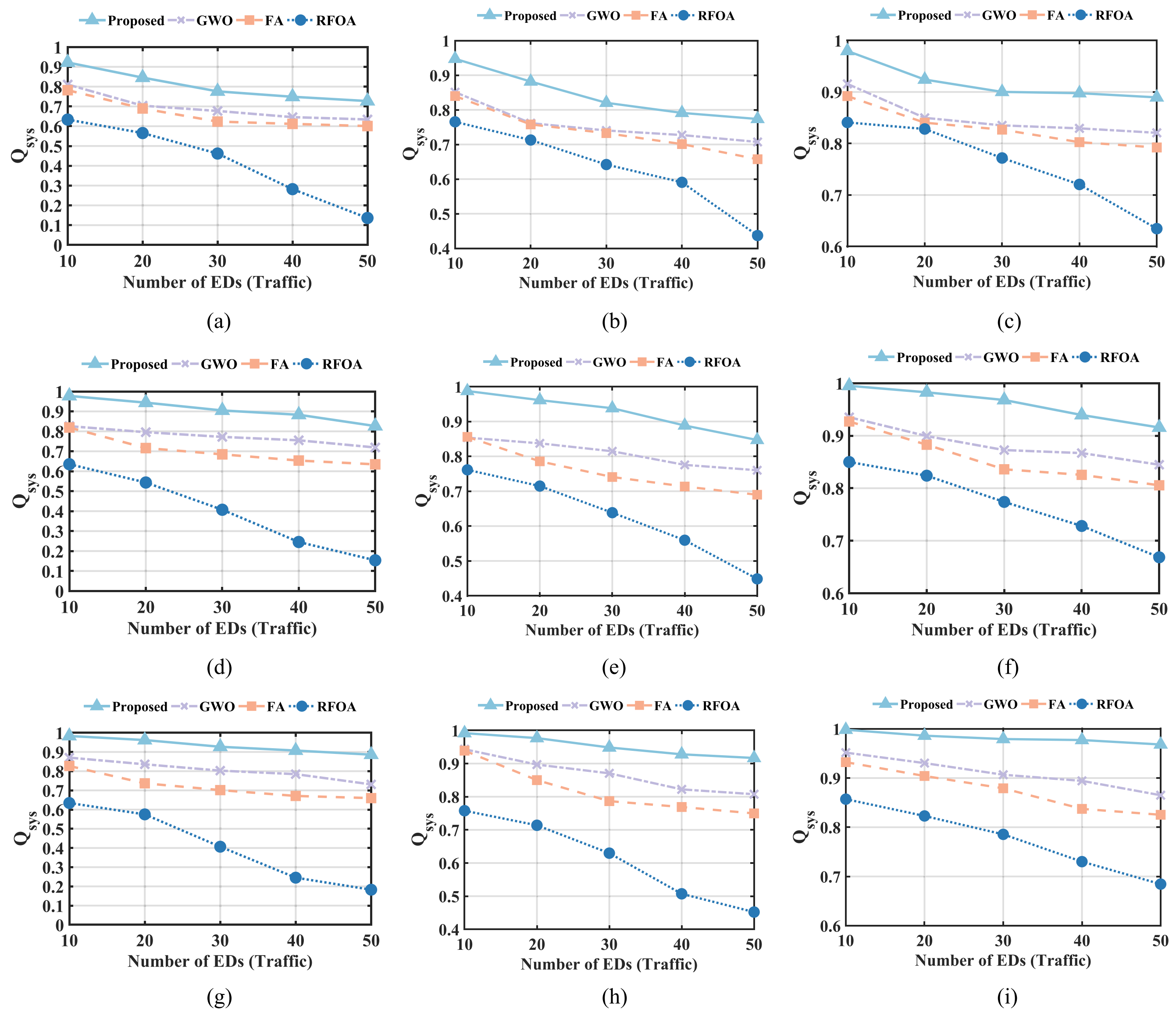


Fig. 4. Utility function values with different number of EDs for varying numbers of ESs and weight coefficients. (a) $M = 2$, $\gamma = 0.8$. (b) $M = 2$, $\gamma = 0.5$. (c) $M = 2$, $\gamma = 0.2$. (d) $M = 4$, $\gamma = 0.8$. (e) $M = 4$, $\gamma = 0.5$. (f) $M = 4$, $\gamma = 0.2$. (g) $M = 6$, $\gamma = 0.8$. (h) $M = 6$, $\gamma = 0.5$. (i) $M = 6$, $\gamma = 0.2$.

as the number of EDs increases incrementally from 5 to 50 in steps of 5. The weight parameter γ is set to 0.5, and task deadlines $d_{i,k}$ range from 0.7 to 1.1 times the local completion time. Simulations are conducted, with results depicted in Fig. 5. Initially, with a small number of EDs, resources are sufficient, allowing all algorithms to complete tasks within their deadlines at a 100% success rate. However, as the number of EDs increases, resource contention escalates, leading to time constraint violations in some cases. Notably, when other algorithms fail to maintain a 100% success rate, the proposed method continues to perform flawlessly up to 35 EDs. Beyond 35 EDs, no algorithm achieves 100% success. Among the algorithms, RFOA yields the worst results, while FA and GWO maintain relatively high-success rates. The proposed method consistently outperforms all others, showing improvements of up to 12.56% over GWO, 17.46% over FA, and 32.39% over RFOA in the most resource-constrained scenarios when the ED number approaches 50. These results demonstrate that

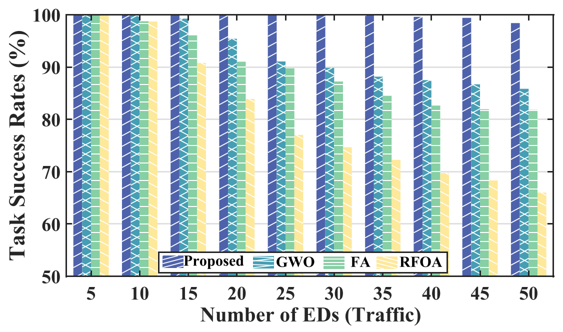


Fig. 5. Task execution success rates under varying EDs.

the proposed method excels in identifying an optimal task offloading strategy across various conditions.

3) *AoI-Oriented Versus Latency-Oriented:* To validate the superiority of the AoI-oriented offloading optimization algorithm over its latency-oriented counterpart, we conduct

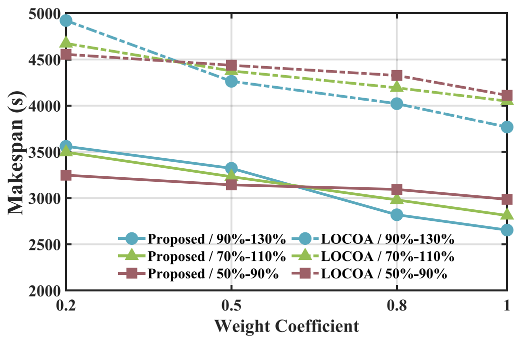


Fig. 6. Makespan under different number weight coefficient varying with task deadline constraints.

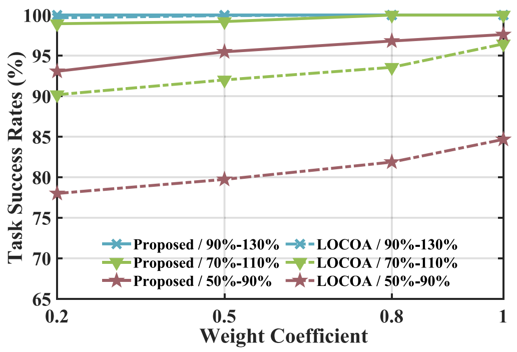


Fig. 7. Task execution success rates under different number weight coefficient varying with task deadline constraints.

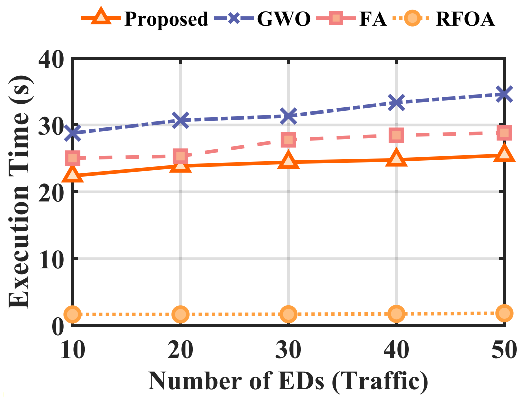


Fig. 8. Execution time comparison to other algorithms.

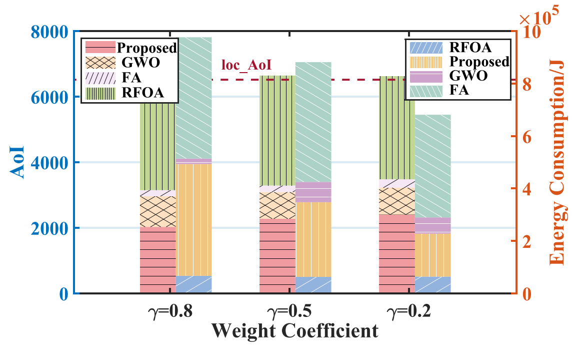


Fig. 9. AoI and energy consumption of the proposed algorithm, GWO, FA, and RFOA for executing all tasks on the real platforms. "loc_AoI" denotes the AoI value of task execution entirely on local devices.

a comparative analysis between the proposed method and LOCOA, which serves as a representative of latency-oriented optimization. The comparison focuses on two key aspects: total task completion latency and task success rate. To facilitate subsequent discussions, we will alternately use "AoI-oriented algorithm" and "the proposed method," as well as "latency-oriented algorithm"'and "LOCOA," depending on the context.

Here, we configure 2 ESs and 30 EDs. Additionally, three sets of task deadline constraints $d_{i,k}$ are established, corresponding to 0.5–0.9, 0.7–1.1, and 0.9–1.3 times the local completion time of tasks. As shown in Figs. 6 and 7, the results indicate that when the deadline constraints are relatively lenient, the proposed method consistently achieves lower makespan across all weight settings compared to LOCOA, with an average reduction of 27.30%. Moreover, the task success rate of the proposed method remains at 100%, while LOCOA experiences instances where some tasks fail to meet their deadlines under lower latency optimization weights.

As the task deadlines become slightly tighter, the advantage of the AoI-oriented algorithm in terms of makespan becomes more evident, with an average reduction of 28.85%. Although some task execution failures may occur at this stage, the overall success rate of the proposed method remains higher than that of LOCOA, with an average improvement of 6.69%. Notably, a 100% success rate is achieved when the AoI optimization weight is set to 1 and 0.8. Further tightening the deadline constraints leads to a remarkable reduction of 30.56% in total task completion latency for the AoI-oriented algorithm compared to the latency-oriented algorithm, with maximum improvements in task success rates reaching up to 15.73% and an average improvement of 14.67%.

These findings highlight the superior performance of the AoI-oriented offloading optimization scheme regarding both total task completion time and task success rates under varying deadline constraints. This is because AoI-oriented optimization can better perceive situational changes, meeting the real-time requirements of devices.

4) *Overhead:* As depicted in Fig. 8, the execution time of RFOA remains low and grows gently with the number of EDs due to its random offloading without complex optimization. In contrast, the proposed algorithm, along with the GWO and FA algorithms, relies on iterative optimization to derive the final solution, inherently consuming additional computational resources and time. Notably, our proposed algorithm consistently outperforms GWO and FA in terms of execution time, and despite its reliance on iterative optimization, its execution time is still within a reasonable range. In AoI-based scenario applications, such as smart agriculture, logistics tracking, and telemedicine, this execution time is completely acceptable, fully demonstrating its good practical application feasibility.

C. Experimental Results

To be more convincing, we implement our proposed SPSO-based method and conduct experimental tests on NVIDIA platforms. Fig. 9 compares the AoI and energy consumption

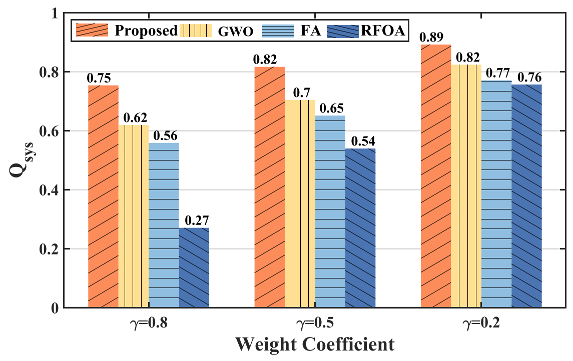


Fig. 10. Utility function values with different weight coefficients under real testbed platform.

values obtained by completing the tasks for all EDs using the proposed method, GWO, FA, and RFOA in case the weight coefficient value is 0.8, 0.5, and 0.2, respectively. The results clearly indicate that the proposed method achieves reduced AoI and energy consumption across various weight parameter settings compared to GWO and FA. Moreover, as the weight coefficient decreases, the system gradually shifts from optimizing AoI to optimizing energy consumption. Specifically, with decreasing the weight coefficient, the AoI values show an increasing trend, while the energy consumption values exhibit a decreasing trend. It is also easily noticed that RFOA has a significantly lower energy consumption than the other three algorithms because all tasks are offloaded to the edge or cloud server for execution, leaving only the energy consumption for task transmission on the EDs.

However, the significant reduction in energy consumption comes at the expense of higher AoI values. As shown in Fig. 9, the value of AoI of RFOA even surpasses that of loc_AoI, as all tasks require offloading, intensifying competition for communication and computing resources, inevitably leading to prolonged task offloading and execution latency along with a substantial increase in AoI. Consequently, it can be concluded that although the resources of EDs in the EECC architecture are limited, they should be utilized wisely. Only through the collaborative processing among the end, edge, and cloud, the desirable outcomes can be achieved.

We also compare the utility values of each group of experiments in Fig. 10. The utility value of the proposed method consistently surpasses those of other algorithms across various weight coefficient settings in the final results, aligning with the simulation results. Specifically, when γ is 0.8, the proposed method surpasses GWO by 13.46%, FA by 19.51%, and markedly outperforms RFOA by a significant 48.29%. When γ is 0.5, the proposed method exceeds GWO by 11.24%, outperforms FA by 16.52%, and remarkably surpasses RFOA by 27.70%. When γ is 0.2, the proposed method achieves a 6.82% increase compared to GWO, a 12.08% improvement over FA, and a 13.54% enhancement in comparison to RFOA.

VII. Conclusion

Achieving efficient computation offloading and resource allocation within EECC environments is fraught with challenges, such as system intricacy, resource heterogeneity, latency sensitivity, and energy management. Given that AoI can directly quantify information freshness, offering a more holistic and timely evaluation of system performance than conventional latency measures, thereby enhancing decision-making quality, we delve into the intricate joint optimization of task offloading and resource allocation in EECC systems, targeting the minimization of AoI and energy consumption under constraints of deadlines, bandwidth, and computation capacity. Specifically, we first construct the mathematical relationship between task offloading decisions and AoI, and then formulate the optimization problem as a MINLP problem. To address this problem, we devise a solution based on SPSO, which partitions the decision space into multiple nonintersecting decision areas tailored to the problem's features. We customize the position, velocity, updating rules, and fitness function of SPSO, and propose a two-phase computation offloading and resource allocation algorithm. Finally, extensive simulation-based and testbed results demonstrate that the proposed method significantly outperforms existing algorithms both in AoI and energy consumption. To support reproducibility and facilitate further research, the source code of the proposed framework has been made publicly available at https://github.com/ghost601010/AoI_TORA_exp/tree/master/project.

References

[1] "The state of mobile Internet connectivity," Accessed: Oct. 2023. [Online]. Available: https://www.gsma.com/r/wp-content/uploads/2023/10/The-State-of-Mobile-Internet-Connectivity-Report-2023.pdf/

[2] X. Han, H. Pan, Z. Wang, and J. Li, "Successive interference cancellation-enabled timely status update in linear multi-hop wireless networks," *IEEE Trans. Mobile Comput.*, vol. 24, no. 6, pp. 5298–5311, Jun. 2025, doi: 10.1109/TMC.2025.3529462.

[3] A. Ndikumana, K. K. Nguyen, and M. Cheriet, "Age of processing-aware offloading decision for autonomous vehicles in 5G open RAN environment," *IEEE Trans. Mobile Comput.*, vol. 23, no. 7, pp. 7865–7877, Jul. 2024.

[4] X. Song, X. Qin, Y. Tao, B. Liu, and P. Zhang, "Age based task scheduling and computation offloading in mobile-edge computing systems," in *Proc. IEEE WCNCW*, 2019, pp. 1–6.

[5] Q. Chen, S. Guo, Z. Cai, J. Li, T. Shi, and H. Gao, "Peak AoI minimization at wireless-powered network edge: From the perspective of both charging and transmitting," *IEEE/ACM Trans. Netw.*, vol. 32, no 1, pp. 806–821, Feb. 2024.

[6] Y. Ding, K. Li, C. Liu, and K. Li, "A potential game theoretic approach to computation offloading strategy optimization in end–edge–cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 6, pp. 1503–1519, Jun. 2022.

[7] C. Kai, H. Zhou, Y. Yi, and W. Huang, "Collaborative cloud-edge-end task offloading in mobile-edge computing networks with limited communication capability," *IEEE Trans. Cogn. Commun. Netw.*, vol. 7, no. 2, pp. 624–634, Jun. 2021.

[8] Y. Zeng et al., "ExpertDRL: Request dispatching and instance configuration for serverless edge inference with foundation models," *IEEE Trans. Mobile Comput.*, vol. 24, no. 9, pp. 8089–8104, Sep. 2025, doi: 10.1109/TMC.2025.3553201.

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

[10] H. Djigal, J. Xu, L. Liu, and Y. Zhang, "Machine and deep learning for resource allocation in multi-access edge computing: A survey," *IEEE Commun. Surveys Tuts.*, vol. 24, no. 4, pp. 2449–2494, 4th Quart., 2022.

[11] H. Zhou, M. Li, P. Sun, B. Guo, and Z. Yu, "Accelerating federated learning via parameter selection and pre-synchronization in mobile edge-cloud networks," *IEEE Trans. Mobile Comput.*, vol. 23, no. 11, pp. 10313–10328, Nov. 2024.

[12] H. Zhou, H. Wang, Z. Yu, G. Bin, M. Xiao, and J. Wu, "Federated distributed deep reinforcement learning for recommendation-enabled edge caching," *IEEE Trans. Services Comput.*, vol. 17, no. 6, pp. 3640–3656, Nov./Dec. 2024.

[13] H. She, L. Yan, and Y. Guo, "Efficient end–edge–cloud task offloading in 6g networks based on multiagent deep reinforcement learning," *IEEE Internet Things J.*, vol. 11, no. 11, pp. 20260–20270, Jun. 2024.

[14] X. Chen, J. Zhang, B. Lin, Z. Chen, K. Wolter, and G. Min, "Energy-efficient offloading for DNN-based smart IoT systems in cloud-edge environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 3, pp. 683–697, Mar. 2022.

[15] H. Xiao, J. Huang, Z. Hu, M. Zheng, and K. Li, "Collaborative cloud-edge-end task offloading in MEC-based small cell networks with distributed wireless backhaul," *IEEE Trans. Netw. Service Manag.*, vol. 20, no. 4, pp. 4542–4557, Dec. 2023.

[16] Y. Shi, C. Yi, R. Wang, Q. Wu, B. Chen, and J. Cai, "Service migration or task rerouting: A two-timescale online resource optimization for MEC," *IEEE Trans. Wireless Commun.*, vol. 23, no. 2, pp. 1503–1519, Feb. 2024.

[17] C. Xu, Q. Xu, J. Wang, K. Wu, K. Lu, and C. Qiao, "AoI-centric task scheduling for autonomous driving systems," in *Proc. IEEE INFOCOM*, 2022, pp. 1019–1028.

[18] R. Li, Q. Ma, J. Gong, Z. Zhou, and X. Chen, "Age of processing: Age-driven status sampling and processing offloading for edge-computing-enabled real-time IoT applications," *IEEE Internet Things J.*, vol. 8, no. 19, pp. 14471–14484, Oct. 2021.

[19] Q. Chen, S. Guo, W. Xu, Z. Cai, L. Cheng, and H. Gao, "AoI minimization charging at wireless-powered network edge," in *Proc. IEEE ICDCS*, 2022, pp. 713–723.

[20] F. Song, Q. Yang, M. Deng, H. Xing, Y. Liu, and X. Yu, "AoI and energy tradeoff for aerial-ground collaborative MEC: A multi-objective learning approach," *IEEE Trans. Mobile Comput.*, vol. 23, no. 12, pp. 11278–11294, Dec. 2024.

[21] Y. Yang, T. Song, J. Yang, H. Xu, and S. Xing, "Joint energy and AoI optimization in UAV-assisted MEC-WET systems," *IEEE Sensors J.*, vol. 24, no. 9, pp. 15110–15124, May 2024.

[22] Q. Wang, X. Liang, H. Zhang, and L. Ge, "AoI-aware energy efficiency resource allocation for integrated satellite-terrestrial IoT networks," *IEEE Trans. Green Commun. Netw.*, vol. 9, no. 1, pp. 125–139, Mar. 2025.

[23] J. Bi, Z. Wang, H. Yuan, J. Zhang, and M. Zhou, "Cost-minimized computation offloading and user association in hybrid cloud and edge computing," *IEEE Internet Things J.*, vol. 11, no. 9, pp. 16672–16683, May 2024.

[24] Z. Xiao, J. Shu, H. Jiang, G. Min, H. Chen, and Z. Han, "Perception task offloading with collaborative computation for autonomous driving," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 2, pp. 457–473, Feb. 2023.

[25] X. Zhai, Y. Peng, and X. Guo, "Edge-cloud collaboration for low-latency, low-carbon, and cost-efficient operations," *Comput. Elect. Eng.*, vol. 120, Dec. 2024, Art. no. 109758.

[26] X. Qu and H. Wang, "Emergency task offloading strategy based on cloud-edge-end collaboration for smart factories," *Comput. Netw.*, vol. 234, Oct. 2023, Art. no. 109915.

[27] T. Liu, L. Fang, Y. Zhu, W. Tong, and Y. Yang, "A near-optimal approach for online task offloading and resource allocation in edge-cloud orchestrated computing," *IEEE Trans. Mobile Comput.*, vol. 21, no. 8, pp. 2687–2700, Aug. 2022.

[28] T. Tang, C. Li, and F. Liu, "Collaborative cloud-edge-end task offloading with task dependency based on deep reinforcement learning," *Comput. Commun.*, vol. 209, pp. 78–90, Sep. 2023.

[29] L. Zhai, Z. Lu, J. Sun, and X. Li, "Joint task offloading and computing resource allocation with DQN for task-dependency in multi-access edge computing," *Comput. Netw.*, vol. 263, May 2025, Art. no. 111222.

[30] Z. Niu, H. Liu, Y. Ge, and J. Du, "Distributed hybrid task offloading in mobile-edge computing: A potential game scheme," *IEEE Internet Things J.*, vol. 11, no. 10, pp. 18698–18710, May 2024.

[31] X. Ma, A. Zhou, Q. Sun, and S. Wang, "Freshness-aware information update and computation offloading in mobile-edge computing," *IEEE Internet Things J.*, vol. 8, no. 16, pp. 13115–13125, Aug. 2021.

[32] J. Huang, H. Gao, S. Wan, and Y. Chen, "AoI-aware energy control and computation offloading for industrial IoT," *Future Gener. Comput. Syst.*, vol. 139, pp. 29–37, Feb. 2023.

[33] Q. Chen, Z. Cai, L. Cheng, F. Wang, and H. Gao, "Joint near-optimal age-based data transmission and energy replenishment scheduling at wireless-powered network edge," in *Proc. IEEE INFOCOM*, 2022, pp. 770–779.

[34] C. Yi, J. Cai, and Z. Su, "A multi-user mobile computation offloading and transmission scheduling mechanism for delay-sensitive applications," *IEEE Trans. Mobile Comput.*, vol. 19, no. 1, pp. 29–43, Jan. 2020.

[35] J. Zhou, X. Hou, Y. Zeng, P. Cong, W. Jiang, and S. Guo, "Quality of experience and reliability-aware task offloading and scheduling for multi-user mobile-edge computing systems," *IEEE Trans. Services Comput.*, vol. 18, no. 3, pp. 1683–1696, May/Jun. 2025.

[36] J. Kwak, Y. Kim, J. Lee, and S. Chong, "DREAM: Dynamic resource and task allocation for energy minimization in mobile cloud systems," *IEEE J. Sel. Areas Commun.*, vol. 33, no. 12, pp. 2510–2523, Dec. 2015.

[37] Z. Ji, S. Wu, and C. Jiang, "Cooperative multi-agent deep reinforcement learning for computation offloading in digital twin satellite edge networks," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 11, pp. 3414–3429, Nov. 2023.

[38] Y. Zhang, Y. Liu, J. Zhou, J. Sun, and K. Li, "Slow-movement particle swarm optimization algorithms for scheduling security-critical tasks in resource-limited mobile edge computing," *Future Gener. Comput. Syst.*, vol. 112, pp. 148–161, Nov. 2020.

[39] B. Aygun, B. G. Kilic, N. Arici, A. Cosar, and B. Tuncsiper, "Application of binary PSO for public cloud resources allocation system of Video on Demand (VoD) services," *Appl. Soft Comput.*, vol. 99, Feb. 2021, Art. no. 106870.

[40] W. Ren and X. Wu, "A modified simple particle swarm optimization using dynamically changing learning factor," *Techn. Autom. Appl.*, vol. 31, no. 10, pp. 9–11, 2012.

[41] S. Choudhary, S. Sugumaran, A. Belazi, and A. A. A. El-Latif, "Linearly decreasing inertia weight PSO and improved weight factor-based clustering algorithm for wireless sensor networks," *J. Ambient Intell. Humanized Comput.*, vol. 14, pp. 6661–6679, Jun. 2023.

[42] W. Jiang, J. Zhou, P. Cong, G. Zhang, and S. Hu, "QoE and reliability-aware task scheduling for multi-user mobile-edge computing," in *Proc. WASA*, 2022, pp. 380–392.

[43] X. Chen, "Decentralized computation offloading game for mobile cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 4, pp. 974–983, Apr. 2015.

[44] M. Yu, A. Liu, N. N. Xiong, and T. Wang, "An intelligent game-based offloading scheme for maximizing benefits of IoT-edge-cloud ecosystems," *IEEE Internet Things J.*, vol. 9, no. 8, pp. 5600–5616, Apr. 2022.

[45] S. Jošilo and G. Dán, "A game theoretic analysis of selfish mobile computation offloading," in *Proc. IEEE INFOCOM*, 2017, pp. 1–9.

[46] Y. Sun et al., "Accurate and rapid CT image segmentation of the eyes and surrounding organs for precise radiotherapy," *Med. Phys.*, vol. 46, no. 5, pp. 2214–2222, 2019.

[47] M. J. F. Calero, M. Aldás, J. Lázaro, A. Gardel, N. Onofa, and B. Quinga, "Pedestrian detection under partial occlusion by using logic inference, HOG and SVM," *IEEE Latin America Trans.*, vol. 17, no. 9, pp. 1552–1559, Sep. 2019.

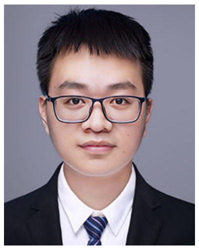
[48] X. Lv, "Cifar-10 image classification based on convolutional neural network," *Front. Signal Process.*, vol. 4, no. 4, pp. 100–106, 2020.

[49] S. Zhang, Z. Wei, Y. Wang, and T. Liao, "Sentiment analysis of chinese micro-blog text based on extended sentiment dictionary," *Future Gener. Comput. Syst.*, vol. 81, pp. 395–403, Apr. 2018.

[50] S. Lin, L. Li, J. Chen, P. Cong, T. Wang, and J. Zhou, "IATS: Information-age aware task scheduling for vehicle-road-cloud cooperative systems," *J. Syst. Archit.*, vol. 167, Oct. 2025, Art. no. 103480.

Youling Zeng received the B.S. degree in computer science and technology from Nanjing University of Science and Technology, Nanjing, China, in 2023, where she is currently pursuing the M.S. degree.

Her research interests are in the area of edge computing and caching.

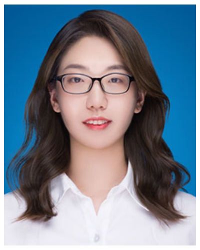


Yue Zeng received the Ph.D. degree from Nanjing University, Nanjing, China, in 2023.

He is currently an Associate Professor with Nanjing University of Science and Technology, Nanjing. He has published more than a dozen papers in top journals and conferences, including IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON SERVICES COMPUTING, IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE TRANSACTIONS ON COMMUNICATIONS, IEEE TRANSACTIONS ON CLOUD COMPUTING and IEEE CVPR. His research interests include edge intelligence, deep reinforcement learning, machine learning training and inference, federated learning, and serverless computing.

Jining Chen received the B.S. and M.S. degrees from Guangxi University, Nanning, China.

He is a Senior Engineer with the Digital Infrastructure Key Laboratory, Guangxi Zhuang Autonomous Region Information Center, Nanning, China. He has published more than ten academic papers, including IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY. His research interests are in the area of edge computing and end–edge–cloud.



Yufan Shen received the B.S. degree in management from Ningbo University, Ningbo, China, in 2020. She is currently pursuing the Ph.D. degree with Nanjing University of Science and Technology, Nanjing, China.

Her research interests include edge computing and IoT.

Liying Li received the Ph.D. degree from the Department of Computer Science and Technology, East China Normal University, Shanghai, China, in 2022.

She is currently an Assistant Professor with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China. Her current research interests are in the areas of cyber–physical systems, IoT resource management, data mining, and distributed artificial intelligence.

Peijin Cong received the B.S. and Ph.D. degrees in computer science from East China Normal University, Shanghai, China, in 2016 and 2021, respectively.

She is currently an Associate Professor with the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China. She has published 30 refereed papers. Her research interests are in the areas of cloud computing, service computing, and IoT.

Junlong Zhou (Member, IEEE) received the Ph.D. degree in computer science from East China Normal University, Shanghai, China, in 2017.

He was a Visiting Scholar with the University of Notre Dame, Notre Dame, IN, USA, from 2014 to 2015. He is currently an Associate Professor with Nanjing University of Science and Technology, Nanjing, China. He has published 120 refereed papers, including more than 40 in premier IEEE/ACM Transactions. His research interests include edge computing, cloud computing, and embedded systems.

Dr. Zhou received the Best Paper Awards from IEEE iThings 2020, IEEE CPSCom 2022, and IEEE ICITES 2024. He has been an Associate Editor of the *Sustainable Computing: Informatics and Systems*, the *Journal of Circuits, Systems, and Computers*, and *IET Cyber–Physical Systems: Theory & Applications*, and a Subject Area Editor of the *Journal of Systems Architecture*.

Keqin Li (Fellow, IEEE) received the B.S. degree in computer science from Tsinghua University, Beijing, China, in 1985, and the Ph.D. degree in computer science from the University of Houston, Houston, TX, USA, in 1990.

He is a SUNY Distinguished Professor with the State University of New York, New Paltz, NY, USA, and a National Distinguished Professor with Hunan University, Changsha, China. He has authored or co-authored more than 1130 journal articles, book chapters, and refereed conference papers. He holds nearly 80 patents announced or authorized by the Chinese National Intellectual Property Administration.

Dr. Li 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 won the IEEE Region 1 Technological Innovation Award (Academic) in 2023. He was a recipient of the 2022–2023 International Science and Technology Cooperation Award and the 2023 Xiaoxiang Friendship Award of Hunan Province, China. Since 2020, he has been among the world's top few most influential scientists in parallel and distributed computing regarding single-year impact (ranked #2) and career-long impact (ranked #4) based on a composite indicator of the Scopus citation database. He is listed in Scilit Top Cited Scholars (2023–2024) and is among the top 0.02% out of over 20 million scholars worldwide based on top-cited publications. He is listed in ScholarGPS Highly Ranked Scholars (2022–2024) and is among the top 0.002% out of over 30 million scholars worldwide based on a composite score of three ranking metrics for research productivity, impact, and quality in the recent five years. He is a member of the SUNY Distinguished Academy. He is an AAAS Fellow, an AAIA Fellow, an ACIS Fellow, and an AIIA Fellow. He is a member of the European Academy of Sciences and Arts. He is a member of Academia Europaea (Academician of the Academy of Europe).