Journal of Systems Architecture 167 (2025) 103469

Contents lists available at ScienceDirect

Journal of Systems Architecture

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

Dynamic task offloading and resource allocation for energy-harvesting end–edge–cloud computing systems☆

Xiaozhu Song [a], Qianpiao Ma [a], Zheng Gan [b], Liying Li [a], Peijin Cong [a], Junlong Zhou [a,*]

[a] School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, 210094, China
[b] Guangxi Key Laboratory of Digital Infrastructure, Guangxi Zhuang Autonomous Region Information Center, Nanning, 530000, China

ARTICLE INFO

Keywords:
End–edge–cloud computing
Energy harvesting systems
Task offloading
Resource allocation
Execution delay
Reliability

ABSTRACT

In end–edge–cloud (EEC) computing, end devices (EDs) offload compute-intensive tasks to nearby edge servers or the cloud server to alleviate processing burdens and enable a flexible computing architecture. However, resource constraints and dynamic environments pose significant challenges for EEC task offloading and resource allocation, including real-time requirements, unreliable task execution, and limited battery energy, especially in energy harvesting (EH) systems, in which battery energy remains unstable due to its inherent fluctuations. Existing task offloading and resource allocation approaches often fail to address these challenges holistically, leading to degraded performance and potential task execution failures. In this paper, we propose a novel task offloading and resource allocation method for EH EEC computing, aiming to optimize long-term performance by minimizing delay and energy consumption while ensuring task execution reliability and battery energy stability. Specifically, we formulate task offloading and resource allocation as a cost optimization problem under constraints such as ED capacity, task reliability, and energy consumption. To solve this problem, we first leverage Lyapunov optimization to decouple the original time-dependent problem. Then we derive optimal closed-form solutions for computation and transmission power resource allocation. Based on these solutions, we propose a multiple discrete particle swarm optimization algorithm to determine task offloading decision. Extensive experiments demonstrate the superiority of our method in balancing delay, execution reliability, and energy stability under varying conditions.

1. Introduction

With the development of the Internet of Things (IoT), massive amounts of data are generated by IoT devices. According to IDC's forecast, the generated data will reach 159.2 ZB in 2024 and is expected to more than double by 2028, reaching 384.6 ZB, with a compound annual growth rate of 24.4% [1]. End devices (EDs) have become an indispensable part of people's daily lives and are expected to process the rapidly increasing data. However, EDs typically use processors and batteries with limited capacity [2]. Processing all generated data on these devices can be time- and energy-consuming. To this end, *task offloading* technology is proposed to alleviate the processing burden on EDs, where the compute-intensive tasks are offloaded to powerful remote computing platforms for processing [3].

Currently, task offloading solutions can be categorized into three main types. The most traditional solution for task offloading is based on cloud computing, *i.e.*, **end-cloud offloading**, where tasks are offloaded to the cloud server for remote execution. The powerful processing capabilities of cloud servers enable them to execute offloaded tasks rapidly [4]. However, the distance between cloud servers and EDs may cause network congestion and increased latency, negatively affecting the execution of tasks for real-time applications [5]. To address these issues, **end-edge offloading** [6,7] which is developed from edge computing, has emerged as an alternative solution, leveraging nearby edge servers for task execution. This proximity enables shorter response times than the end-cloud offloading solution, making it ideal for applications requiring real-time task processing. Nevertheless, the computation resources of edge servers are relatively limited [8], which

☆ This work was supported in part by the National Natural Science Foundation of China under Grants 62302221, 62302216, 62172224, 62402537, in part by the Natural Science Foundation of Jiangsu Province under Grants BK20230913, BK20230912, BK20220138, in part by the Fundamental Research Funds for the Central Universities under Grants 30922010318, 30924010815, 30925010408, 30925010515, and in part by the Open Project Program of Guangxi Key Laboratory of Digital Infrastructure (Grant Number: GXDIOP2024006).

* Corresponding author.
E-mail addresses: songxiaozhu@njust.edu.cn (X. Song), maqianpiao@njust.edu.cn (Q. Ma), ganzheng@gxi.gov.cn (G. Zheng), liyingli@njust.edu.cn (L. Li), cpj@njust.edu.cn (P. Cong), jlzhou@njust.edu.cn (J. Zhou).

https://doi.org/10.1016/j.sysarc.2025.103469
Received 24 January 2025; Received in revised form 17 April 2025; Accepted 21 May 2025
Available online 7 June 2025

Table 1
Comparison of approaches in EEC.

Approach	Optimize execution delay	Optimize energy consumption	Consider dynamic environments	Consider reliability	Consider Battery Energy Stability
WBS [22]	✓				
MaOPPC [23]	✓				
CCORAM [24]	✓	✓			
IOR [25]	✓	✓			
GDTO [26]	✓	✓			
DMQTO [27]	✓	✓	✓		
SMBO [28]	✓	✓	✓		
TPDRTO [29]	✓	✓	✓		
TSDA [30]	✓	✓	✓		
EAP-OPT [31]			✓	✓	
TOLERANCER [32]			✓	✓	
RES [33]			✓	✓	
OAPCR [34]			✓	✓	
Ours	✓	✓	✓	✓	✓

can lead to long execution times for compute-intensive tasks. **End–edge–cloud (EEC) offloading** [4] combines the benefits of both cloud computing and edge computing by distributing tasks among EDs, edge servers, and the cloud server. This collaboration establishes a flexible framework, meeting the varying task executing requirements of diverse applications, such as delay and energy constraints.

However, there are still some challenges in existing EEC offloading research.

- **Real-time Requirement:** Many ED's applications (*e.g.*, autono mous driving and augmented reality) typically have stringent timing constraints that require them to be completed before their deadlines. In the EEC system, task offloading needs to guarantee low latency to ensure the real-time requirement [9–11]. However, the distributed and multi-tiered nature of EEC architecture makes it difficult to achieve consistently low delay, since data traverses multiple nodes, each with varying network conditions and workloads.
- **Unreliable Task Execution:** In the dynamic environments of EEC computing, task execution reliability is frequently impacted by factors like resource competition and performance fluctuations [12–14]. This often decreases the probability of successfully completing tasks without suffering transient failures, especially during peak usage periods.
- **Limited Battery Energy:** EDs such as robots, cameras, and portable weather stations typically rely on batteries with limited capacity for power, leading to potential energy shortages during periods of high-demand operation [15–18]. Some studies focus on energy harvesting (EH) techniques to capture renewable resources from the environment and convert them into battery energy [19–21]. However, the inherent randomness in EH can cause fluctuations in the ED's energy supply. For example, robots may suspend their mission execution in low energy conditions, which reduces task execution efficiency and potentially shortens battery lifespan.

In recent years, a wide range of approaches for EEC computing have been proposed to address the above challenges. A comprehensive comparison of these approaches is summarized in Table 1. For example, [22,23] explore the collaborative task offloading methods aimed at reducing delays, while [24,25] propose methods that jointly optimize delay and energy consumption. However, these methods are primarily designed for static scenarios and face significant limitations in dynamic scenarios. To address this, [27–29] propose offloading methods tailored for dynamic scenarios. Although these methods are adaptive to dynamic environments, they ignore task execution reliability, potentially leading to slow responses and incomplete execution. Only a few studies [31–34] address reliability issues in dynamic scenarios to ensure reliable task execution. However, these studies mainly focus on enhancing task execution reliability and do not adequately consider the critical factors related to task offloading, such as delay and energy consumption.

Unlike previous studies, this paper solves the delay, energy, and reliability concerns in EEC computing systems simultaneously. Specifically, we propose a dynamic task offloading and resource allocation method for energy harvesting end–edge–cloud (EH EEC) computing systems, aiming to optimize the long-term performance (*i.e.*, minimizing delay and energy consumption) while ensuring task execution reliability and battery energy stability. The main contributions of this paper are summarized as follows.

- We formulate the task offloading problem to minimize cost (including task execution delay and dropping penalty) within the EH-EEC architecture, subject to constraints such as ED capacity, task execution reliability, and energy consumption, to meet the real-time requirement, ensure reliable task execution, as well as manage the limited and unstable battery energy.
- We design an efficient algorithm to solve the formulated task offloading problem. Specifically, we first decompose the original time-coupled problem by leveraging the Lyapunov optimization to address unstable battery energy in the EH architecture. Next, we derive the optimal closed-form solutions for the resource allocation of EDs and edge servers, including computation and transmission power. Based on these solutions, we propose a multiple discrete particle swarm optimization (MDPSO) algorithm to determine the task offloading strategy.
- We validate the efficacy of the proposed method through extensive simulation experiments. The experimental results, with varying parameters (*e.g.*, task generation probabilities and control parameters), show the superiority of the proposed method in terms of balancing delay, task execution reliability, and battery energy stability.

We organize this paper as follows. Section 2 reviews the related work. Section 3 introduces the specific system models and defines the optimization problem based on the models. Section 4 describes our proposed dynamic task offloading and resource allocation algorithms in detail. Section 5 conducts simulation experiments to evaluate the performance of the proposed algorithm from various aspects, followed by the concluding remarks in Section 6.

2. Related work

This section classifies related work on task offloading and resource allocation into two categories: static and dynamic, all within the context of EEC computing.

2.1. Static task offloading and resource allocation in EEC computing

EEC computing is an architecture that effectively integrates edge and cloud computing, providing enhanced flexibility and effectively addressing diverse user requirements. Most research on EEC computing focuses on optimizing task offloading and resource allocation by reducing delay and minimizing energy consumption. For example, Gao *et al.* [22] proposed an auction-based virtual machine allocation mechanism to address the resource allocation problem for deadline-sensitive tasks. Hu *et al.* [23] introduced a knowledge-mining-based multi-objective evolutionary algorithm to reduce the delay in content popularity prediction tasks. Zhou *et al.* [24] proposed a joint optimization method for computation offloading and service caching in an edge computing-based smart grid, aiming to minimize the task execution time and energy consumption. Xiao *et al.* [25] developed an iterative task offloading method to minimize energy consumption under given delay constraints. Chen *et al.* [26] designed a game-based decentralized approach that maximizes the quality of user experience by optimizing resource allocation. However, these methods often assume static conditions, making them less effective in adapting to dynamic scenarios.

2.2. Dynamic task offloading and resource allocation in EEC computing

To address the various challenges posed by dynamic environments, several studies have designed task offloading and resource allocation methods for this scenario. For instance, Sharma *et al.* [27] proposed a deep meta-reinforcement learning approach to address the multi-task offloading problem in EEC systems. Yuan *et al.* [28] presented a simulated annealing-based migratory bird algorithm for dynamic task offloading. Tang *et al.* [29] designed an offloading algorithm that incorporates task prioritization and deep reinforcement learning to optimize delay and energy consumption jointly. Fan *et al.* [30] developed a time-sliced method for task offloading and resource allocation, effectively minimizing task execution delay under energy consumption constraints. Unlike [27–30], Chen *et al.* [31] investigated the joint optimization problem of coverage and reliability, in which they proposed an optimal and approximate solution based on an integer programming method. Al-Dulaimy *et al.* [32] addressed software and hardware failures by monitoring node states. Qiu *et al.* [33] considered the time-varying faults and proposed an online approximation method to ensure fault tolerance in deploying virtual network functions (VNFs). Similarly, Li *et al.* [34] enhanced the reliability of VNFs by deploying both primary and backup instances.

However, the aspect of green EH in dynamic EEC computing remains unexplored, which provides an innovative solution for EDs that traditionally depend on battery power. In light of the above, we explore the integration of EH technology for dynamic task offloading and resource allocation in EEC computing, focusing on optimizing both delay and energy consumption.

3. System model

In this section, we first introduce the network model, the communication model, the computation model, and the reliability model of our system. Then, we formulate the dynamic task offloading and resource allocation problem in EEC computing. Some important notations in this paper are listed in Table 2.

3.1. Network model

Consider an EEC computing system based on small cell networks. As shown in Fig. 1, the system is divided into three layers: the cloud layer, the edge layer, and the end layer. The cloud layer consists of a high-performance cloud server. The edge layer comprises M small cell networks, each with one edge server. The set of edge servers is denoted as $\mathcal{S} = \{S_m | m \in \{1, 2, \ldots, M\}\}$. Each cell network also deploys a base station to receive and distribute tasks. The end layer consists of N EDs $\mathcal{D} = \{D_n | n \in \{1, 2, \ldots, N\}\}$, which are randomly distributed in different small cell networks. Each ED is equipped with an EH component and fully powered by the collected renewable energy, such as solar energy. The set of EDs within the service range of edge server S_m is denoted as $\mathcal{D}_m$, satisfying $\mathcal{D} = \bigcup_{1 \le m \le M} \mathcal{D}_m$ and $\mathcal{D}_{m_1} \bigcap \mathcal{D}_{m_2} = \emptyset$, $\forall m_1 \neq m_2$.

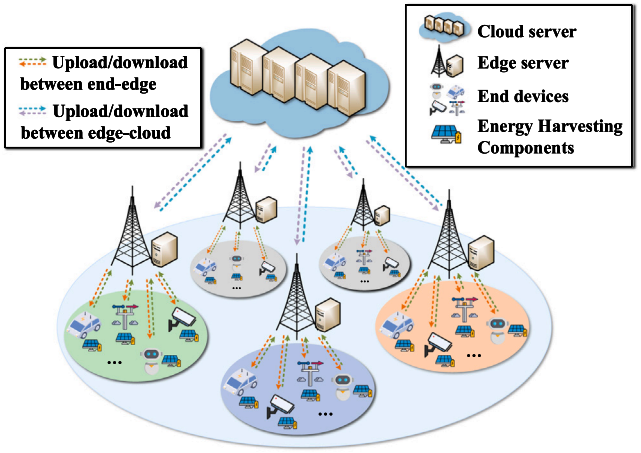


Fig. 1. Architecture of an EEC Computing renewable energy-powered computing system.

Table 2
Key notations.

Symbol	Semantics
$\mathcal{D}$	The set of EDs $\{D_1, D_2, \ldots, D_N\}$
$\mathcal{S}$	The set of edge servers $\{S_1, S_2, \ldots, S_M\}$
$\mathcal{D}_m$	The set of EDs within the range of S_m
τ_n^t	The task generated by D_n at slot t
x_n^t	The offloading decision of τ_n^t
p_n^t	The transmission power of D_n at slot t
f_n^t	The CPU frequency assigned to τ_n^t for execution
N_m^t	The number of tasks received by S_m at slot t
$L_{n,u}^t / L_{n,d}^t$	The end-edge uplink/downlink transmission delay of task τ_n^t
$L_{n,U}^t / L_{n,D}^t$	The edge-cloud uplink/downlink transmission delay of task τ_n^t
$L_{n,\text{comp}}^t$	The computation delay of task τ_n^t
$L_{n,\text{tran}}^t$	The transmission delay of task τ_n^t
E_n^t	The energy consumption of task τ_n^t
R_n^t	The execution reliability of task τ_n^t

To address the uncertainty in energy harvesting, we divide time into discrete time slots $\mathcal{T} = \{0, 1, \ldots\}$, each with a fixed length of Y [20]. At the beginning of each slot, each ED generates a task following an independent and identically distributed (i.i.d.) Bernoulli process [15] with a constant probability $\rho \in [0, 1]$, which is the same value throughout system operation. For instance, when $\rho = 0.3$, it indicates a 30% probability for an ED to generate a task at the beginning of any given time slot. Let ζ_n^t denote the indicator of whether D_n generates a task at slot t or not. Specifically, $\zeta_n^t = 1$ indicates the presence of a task, denoted by τ_n^t, while $\zeta_n^t = 0$ means that no task is generated at slot t. The generated tasks typically have real-time constraints that require them to be completed before their deadlines (*i.e.*, within a time slot). These tasks can be processed locally or offloaded to an edge/cloud server for execution, or they may be dropped if the ED's battery energy is insufficient. We define the task offloading decision as $x_n^t = \{x_{n,l}^t, x_{n,e}^t, x_{n,c}^t, x_{n,d}^t\}^\top$, where $x_{n,l}^t, x_{n,e}^t, x_{n,c}^t, x_{n,d}^t \in \{0, 1\}$. When $x_{n,l}^t = 1$, $x_{n,e}^t = 1$, $x_{n,c}^t = 1$, or $x_{n,d}^t = 1$, it indicates that the task is executed on the ED, edge server, cloud server, or that the task is dropped, respectively. Clearly, the constraint $x_{n,l}^t + x_{n,e}^t + x_{n,c}^t + x_{n,d}^t = \zeta_n^t$ holds for $\forall D_n \in \mathcal{D}$.

3.2. Communication model

When a task is offloaded, it is transmitted to the edge or cloud server via the wireless network. To avoid channel interference, we employ orthogonal frequency division multiple access [9] for wireless communications among EDs, edge servers, and the cloud server.

3.2.1. ED-edge communication

Let $B_{m,u}$ denote the available end-edge uplink bandwidth at edge server S_m. The transmission power of ED D_n for uplink transmission at slot t is represented as p_n^t, and the channel gain between D_n and S_m at slot t is denoted as h_{mn}^t. It is assumed that the wireless channel remains stationary in the same slot and varies among different slots [15]. At each slot t, there are N_m^t tasks uploaded from the EDs in set $\mathcal{D}_m$ to edge server S_m, *i.e.*, $N_m^t = \sum_{D_n \in \mathcal{D}_m} x_{n,e}^t$. As a result, for $\forall D_n \in \mathcal{D}_m$, the achievable uplink transmission rate of task τ_n^t can be given by the Shannon–Hartley formula [35]

$$r_{n,u}^t = \frac{B_{m,u}}{N_m^t} \log_2(1 + \frac{h_{mn}^t p_n^t}{\sigma^2}), \tag{1}$$

where σ^2 is the noise power. Then the uplink transmission delay of task τ_n^t from D_n to S_m can be calculated as

$$L_{n,u}^t = \frac{A_n^t}{r_{n,u}^t}, \tag{2}$$

where A_n^t is the data amount of task τ_n^t. Since the size of the output results is significantly smaller than that of the input data, the downlink energy consumption is excluded from the ED's energy model, as discussed in [36–38]. As a result, the energy consumption of D_n during transmission to the edge server can be expressed as

$$E_{n,u}^t = p_n^t L_{n,u}^t = \frac{p_n^t A_n^t}{r_{n,u}^t}. \tag{3}$$

After task execution at the edge server, the result is sent back to the corresponding ED. Let P_m^t denote the downlink transmission power of edge server S_m. Similarly, the downlink transmission rate $r_{n,d}^t$ from server S_m to ED D_n can also be obtained by the Shannon capacity. Therefore, the downlink transmission delay for the execution result of task τ_n^t from S_m to D_n is

$$L_{n,d}^t = \frac{U_n^t}{r_{n,d}^t}, \tag{4}$$

where U_n^t is the data amount of execution result of task τ_n^t.

3.2.2. Edge-cloud communication

The edge server communicates with the cloud via the wired network [29], where the transmission rate is denoted as $r_{m,U}$. For $\forall D_n \in \mathcal{D}_m$, the uplink transmission delay of task τ_n^t from S_m to the cloud server is

$$L_{n,U}^t = \frac{A_n^t}{r_{m,U}}. \tag{5}$$

The downlink transmission rate from the cloud server to edge server S_m can be denoted as $r_{m,D}$. Once processed by the cloud, the result of τ_n^t is returned to edge server S_m, and the downlink transmission delay is

$$L_{n,D}^t = \frac{U_n^t}{r_{m,D}}. \tag{6}$$

3.3. Computation model

As mentioned earlier, each task can be executed locally on the ED, or offloaded to the edge/cloud for remote execution. We use the Dynamic Voltage and Frequency Scaling (DVFS) technique [39] to adjust the chip voltage and ensure the CPU frequency in a slot. Let $f_{n,l}^t$, $f_{n,e}^t$ and $f_{n,c}^t$ denote the CPU frequency allocated for task τ_n^t when the task is executed at ED, the edge server and the cloud server, respectively. Then the computation delay of τ_n^t can be expressed as

$$L_{n,\text{comp}}^t = \frac{C_n^t}{f_n^t}, \tag{7}$$

where C_n^t is the number of CPU cycles to complete task τ_n^t, which is assumed to be fixed across different execution locations as tasks are executed on CPU cores [25,40,41]. Additionally, f_n^t satisfies

$$f_n^t = \begin{cases} f_{n,l}^t, & x_{n,l}^t = 1 \\ f_{n,e}^t, & x_{n,e}^t = 1 \\ f_{n,c}^t, & x_{n,c}^t = 1 \end{cases}. \tag{8}$$

Specially, if task τ_n^t is executed locally (*i.e.*, $x_{n,l}^t = 1$), the computation energy consumption of D_n is given by

$$E_{n,l}^t = k(f_{n,l}^t)^2 C_n^t, \tag{9}$$

where k is a constant depending on the processor architecture of the ED [15].

3.4. Reliability model

Consider soft faults caused by transient failures during task execution, which generally do not damage the device hardware. According to [13], the initial task execution failure rate for task τ_n^t can be expressed as

$$\mu(f_n^t) = \mu_0 10^{\frac{\delta(f_{\max} - f_n^t)}{f_{\max} - f_{\min}}}, \tag{10}$$

where μ_0 is the fault rate when the device operates at the maximum frequency $f_{\max}$, δ is a constant indicating the sensitivity of the fault rate to voltage scaling, and $f_{\max}/f_{\min}$ is the maximum/minimum processing speed of the system. Let W_n^t denote the vulnerability factor of task τ_n^t [12], which is used to evaluate the reliability of the task. According to the exponential distribution model of faults [42], the execution reliability of task τ_n^t is given as

$$R_n^t = e^{-\mu(f_n^t) W_n^t \frac{C_n^t}{f_n^t}}. \tag{11}$$

Therefore, the system reliability can be calculated as the product of the reliability of all tasks, *i.e.* ,

$$R^t = \prod_{\tau_n^t \in \Gamma} R_n^t. \tag{12}$$

3.5. Problem formulation

As discussed, task τ_n^t can be executed locally, on an edge server, or on the cloud, with corresponding transmission delays in each case. If task τ_n^t is executed locally (*i.e.*, $x_{n,l}^t = 1$), there is no need for offloading, and thus, the transmission delay is zero. If task τ_n^t is offloaded to the associated edge server S_m for execution (*i.e.*, $x_{n,e}^t = 1$), the transmission delay includes both the uplink and downlink transmission delays between D_n and S_m. If task τ_n^t is further offloaded to the cloud server (*i.e.*, $x_{n,c}^t = 1$), the transmission delay includes not only the uplink/downlink transmission delays between D_n and S_m but also the uplink/downlink transmission delays between S_m and the cloud server. Therefore, the total transmission delay of τ_n^t is expressed as

$$L_{n,\text{tran}}^t = \begin{cases} 0, & x_{n,l}^t = 1 \\ L_{n,u}^t + L_{n,d}^t, & x_{n,e}^t = 1 \\ L_{n,u}^t + L_{n,d}^t + L_{n,U}^t + L_{n,D}^t, & x_{n,c}^t = 1 \\ 0, & x_{n,d}^t = 1 \end{cases}. \tag{13}$$

According to Eqs. (7) and (13), the total execution delay L_n^t of task τ_n can be calculated as

$$L_n^t = L_{n,\text{comp}}^t + L_{n,\text{tran}}^t = \begin{cases} \frac{C_n^t}{f_{n,l}^t}, & x_{n,l}^t = 1 \\ \frac{C_n^t}{f_{n,e}^t} + \frac{A_n^t}{r_{n,u}^t} + \frac{U_n^t}{r_{n,d}^t}, & x_{n,e}^t = 1 \\ \frac{C_n^t}{f_{n,c}^t} + \frac{A_n^t}{r_{n,u}^t} + \frac{U_n^t}{r_{n,d}^t} + \frac{A_n^t}{r_{m,U}} + \frac{U_n^t}{r_{m,D}}, & x_{n,c}^t = 1 \\ 0, & x_{n,d}^t = 1 \end{cases}. \tag{14}$$

Since edge and cloud servers typically have abundant energy resources, we focus only on the energy consumption of EDs. If task τ_n is executed locally (*i.e.*, $x_{n,l}^t = 1$), the energy consumption of D_n is equivalent to its local computation energy consumption. In contrast, if τ_n is offloaded to either the edge server or the cloud server (*i.e.*, $x_{n,e}^t = 1$ or $x_{n,c}^t = 1$), the energy consumption of D_n is determined by the energy required for uplink transmission. Therefore, according to Eqs. (3) and (9), the energy consumption of D_n for task τ_n^t can be expressed as

$$E_n^t = \begin{cases} k(f_{n,l}^t)^2 C_n^t, & x_{n,l}^t = 1 \\ \frac{p_n^t A_n^t}{r_{n,u}^t}, & x_{n,e}^t = 1 \\ \frac{p_n^t A_n^t}{r_{n,u}^t}, & x_{n,c}^t = 1 \\ 0, & x_{n,d}^t = 1 \end{cases}. \tag{15}$$

We model the energy harvesting process as a continuous process of arriving energy packets. Let $E_{n,H}^t$ denote the energy collected by D_n during the slot $t-1$, which satisfies the independent homogeneous distribution in different slots and obeys the distribution $E_{n,H}^t \sim U(0, E_H^{\max})$. The portion of $E_{n,H}^t$ stored by D_n is denoted as e_n^t, which is utilized to perform the tasks of slot t. Following [15,19,43,44], we adopt the commonly accepted i.i.d. model to harvest the energy. This model captures the stochastic and intermittent nature of renewable energy processes. Let Q_n^t denote the battery energy of D_n at slot t. Then the variation of battery energy between neighboring slots can be calculated as

$$Q_n^{t+1} = \max\{Q_n^t + e_n^t - E_n^t, 0\}. \tag{16}$$

Task execution delay is widely used to estimate system performance. However, the stochastic and intermittent nature of renewable energy may cause task dropping. Considering both task execution delay and dropping overhead, we define the system execution cost of slot t as

$$\begin{aligned} cost^t &\triangleq \sum_{D_n \in D} cost_n^t \\ &= \sum_{D_n \in D} \mathbf{1}(\zeta_n^t = 1) \left\{ L_n^t + \lambda \cdot \mathbf{1}(x_{n,d}^t = 1) \right\}, \end{aligned} \tag{17}$$

where λ is the penalty for task dropping. Based on the definition, we formulate the dynamic task offloading and resource allocation problem as

$$\mathbb{P}1: \min_{x_n^t, f_n^t, p_n^t, e_n^t} \lim_{T\to\infty} \frac{1}{T} \mathbb{E}\left[\sum_{t=1}^{T} cost^t\right]$$

$$\begin{aligned}
\textbf{s.t.}\ & f_{n,l}^t \le f_l^{\max}, && \forall D_n \in D, t \in \mathcal{T} && \text{(18a)} \\
& \sum_{D_n \in D_m} x_{n,e}^t f_{n,e}^t \le f_{m,e}^{\max}, && \forall S_m \in S, t \in \mathcal{T} && \text{(18b)} \\
& p_n^t \le p_l^{\max}, && \forall D_n \in D, t \in \mathcal{T} && \text{(18c)} \\
& 0 \le E_n^t \le E^{\max}, && \forall D_n \in D, t \in \mathcal{T} && \text{(18d)} \\
& 0 \le e_n^t \le E_{n,H}^t, && \forall D_n \in D, t \in \mathcal{T} && \text{(18e)} \\
& L_n^t \le Y, && \forall D_n \in D, t \in \mathcal{T} && \text{(18f)} \\
& R^t \ge R^{\text{th}}, && t \in \mathcal{T} && \text{(18g)} \\
& x_{n,l}^t + x_{n,e}^t + x_{n,c}^t + x_{n,d}^t = \zeta_n^t, && \forall D_n \in D, t \in \mathcal{T} && \text{(18h)} \\
& x_{n,l}^t, x_{n,e}^t, x_{n,c}^t, x_{n,d}^t \in \{0,1\}, && \forall D_n \in D, t \in \mathcal{T} && \text{(18i)}
\end{aligned}$$

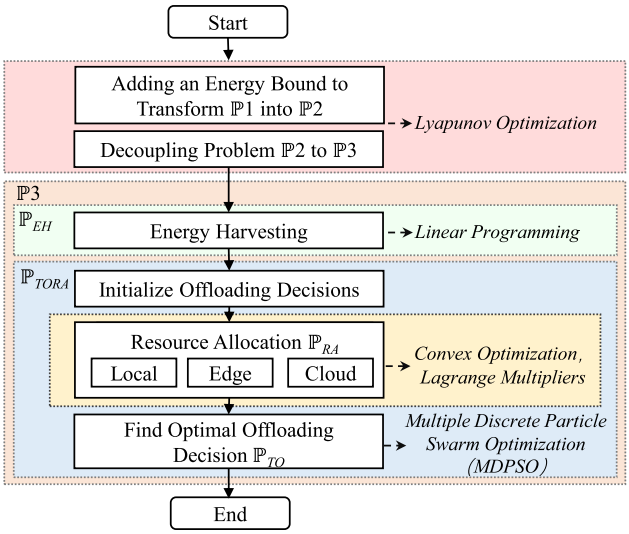


Fig. 2. Overview of our proposed method.

Eq. (18a) indicates that the CPU frequency of each ED cannot exceed its maximum allowed frequency when performing the task locally. Eq. (18b) indicates that the computation resources allocated to the tasks executed on each edge server should be within its capacity. Eq. (18c) ensures that the transmission power of each ED during task offloading remains within its maximum power limit. Eq. (18d) represents that the energy consumption of each ED in slot t does not exceed the maximum discharge threshold of the battery $E^{\max}$. Eq. (18e) indicates that the net energy harvested by the ED at slot t does not exceed the total harvested energy. Eq. (18f) is the deadline constraint for the task. Eq. (18g) denotes that the reliability of executing all tasks in time slot t is not lower than the given threshold R^{th}. Eq. (18h) ensures that tasks can only offload to one location. Eq. (18i) is the binary constraint on the decision variables. Our objective is to determine the task offloading decision x_n^t, the CPU frequency f_n^t, the transmission power p_n^t, and the net energy harvested e_n^t to minimize the long term system execution cost under all the design constraints.

4. Lyapunov-based dynamic task offloading and resource allocation algorithm

4.1. Methodology framework

This section addresses the problem of minimizing execution costs in an EEC computing system and proposes a **D**ynamic **T**ask **O**ffloading and **R**esource **A**llocation (DTORA) framework. The proposed framework contains three key components: Lyapunov-based decoupling, convex optimization-based resource allocation, and multiple discrete particle swarm optimization (MDPSO) based task offloading, as illustrated in Fig. 2. Specifically, to address the time-dependent nature and energy harvesting uncertainties of the original problem $\mathbb{P}1$, we apply Lyapunov optimization by adding a lower energy bound, which ensures that the output energy in each slot is either zero or above this bound [15]. Based on this, we transform the problem $\mathbb{P}1$ into a modified problem $\mathbb{P}2$. According to Lyapunov optimization theory, minimizing the Lyapunov drift plus penalty function enables the transformation of multi-slot coupled constraints into a modified single-slot problem for resolution [19]. Thus, by constructing a Lyapunov drift-plus-penalty function and deriving its upper bound, we ensure the stability of battery energy, which is denoted as $\mathbb{P}3$. Furthermore, we decompose

$\mathbb{P}3$ into two subproblems: the energy harvesting problem $\mathbb{P}_{EH}$ and the task offloading and resource allocation problem $\mathbb{P}_{TORA}$. The $\mathbb{P}_{EH}$ can be solved using linear programming due to the linear property of the objective function of $\mathbb{P}_{EH}$, while the $\mathbb{P}_{TORA}$ can be split into the resource allocation problem $\mathbb{P}_{RA}$ and the task offloading problem $\mathbb{P}_{TO}$. Before solving the resource allocation problem, we first initialize the offloading decision for all tasks. Then we design a convex optimization algorithm to perform resource allocation for the offloaded tasks. This process involves numerical analyses, such as derivatives and Lagrange multipliers, to derive closed-form solutions for the optimal transmission power and frequency resource allocation among local EDs, edge servers, and the cloud server. Lastly, we design the MDPSO algorithm to determine the locations for task execution, which can efficiently find the high-quality task offloading decision that achieves satisfactory performance while meeting all the constraints.

Note that our proposed Lyapunov optimization framework dynamically balances energy supply and demand to stabilize battery levels, ensuring sufficient energy reserves for task execution under varying energy harvesting conditions. If the energy is critically low, task dropping is allowed in our framework to save energy. But our algorithms would adaptively adjust task offloading decisions and local processing strategies to avoid task dropping.

4.2. Decoupling the original problem by Lyapunov optimization

Given the inherently stochastic nature of the energy harvested in each slot, the battery energy levels of EDs fluctuate over time [19]. These fluctuations introduce coupling among system decisions across different slots, complicating the problem-solving process. According to [15], the coupling between decisions in different slots can be mitigated by imposing a battery energy bound. Thus, in our analysis, we introduce a lower bound $E^{\min}$ on the battery output energy for each slot and construct constraint (19a). Specifically, this constraint ensures that the output energy in each slot is either zero or above $E^{\min}$. This modification allows for per-slot optimization without considering historical energy dependencies. The feasibility has been rigorously proved in [15] (Proposition 1). In this way, $\mathbb{P}1$ can be reformulated as follows.

$$\mathbb{P}2: \min_{x_n^t, f_n^t, p_n^t, e_n^t} \lim_{T\to\infty} \frac{1}{T}\mathbb{E}\left[\sum_{t=1}^{T} cost^t\right]$$

s.t. (18a)–(18c), (18e)–(18i)

$$E_n^t \in \{0\} \cup \{E^{\min}, E^{\max}\}. \tag{19a}$$

Next, we address problem $\mathbb{P}2$ using the Lyapunov optimization theory. The standard Lyapunov optimization approach assumes that the operations across different time slots are independent and identically distributed. However, due to the time-dependent nature of system decisions, the direct application of the original Lyapunov optimization method is infeasible. To address this issue, we introduce a weighted perturbation method, which is an effective technique for handling the time dependence of decisions [45]. Specifically, we denote the maximum energy consumed by the ED D_n as

$$\tilde{E}_n^{\max} = \min\left\{\max_{t\in\mathcal{T}}\left\{k(f_{n,l}^t)^2 C_n^t, p_l^{\max} Y\right\}, E^{\max}\right\}. \tag{20}$$

According to [15], we introduce a perturbation parameter θ_n for D_n as a tuning knob to prevent battery energy from hitting boundaries (0 or $E^{\max}$), denoted as

$$\theta_n \geq \tilde{E}_n^{\max} + V\lambda \cdot E^{\min^{-1}}, \tag{21}$$

where V is a weighted control parameter for task execution delay. We define the virtual battery energy queue $\tilde{Q}_n^t$ for the EH device D_n. In Lyapunov optimization, by stabilizing $\tilde{Q}_n^t$ around zero, we can achieve the stability of the actual battery energy Q_n^t. The virtual energy queue $\tilde{Q}_n^t$ can be expressed as

$$\tilde{Q}_n^t = Q_n^t - \theta_n, \tag{22}$$

where Q_n^t is the battery energy as given in Eq. (16). By controlling the virtual energy queue $\tilde{Q}_n^t$ and the control parameter V, our goal is to minimize the Lyapunov drift plus penalty function [19]. This approach can ensure the stability of the energy queue around the designated threshold. Based on this, we build the quadratic Lyapunov function as

$$\mathscr{L}(t) = \frac{1}{2}\sum_{n=1}^{N}(\tilde{Q}_n^t)^2 = \frac{1}{2}\sum_{n=1}^{N}(Q_n^t - \theta_n)^2. \tag{23}$$

Furthermore, the Lyapunov drift function can be derived as

$$\Delta(t) = \mathbb{E}\left[\mathscr{L}(t+1) - \mathscr{L}(t)|\tilde{Q}^t\right], \tag{24}$$

where $\tilde{Q}^t = \left[\tilde{Q}_1^t, \ldots, \tilde{Q}_N^t\right]$ is the set of virtual energy queues for all EDs at slot t. We can obtain the upper bound of $\Delta(t)$ as in the following theorem.

Theorem 1. *There exists an upper bound for $\Delta(t)$, i.e.,*

$$\Delta(t) \leq \Phi + \sum_{n=1}^{N}\tilde{Q}_n^t\mathbb{E}\left[e_n^t - E_n^t|\tilde{Q}^t\right],$$

where $\Phi = \frac{1}{2}\sum_{n=1}^{N}\left[(E_H^{\max})^2 + (E^{\max})^2\right]$.

Proof. Combining Eqs. (16), (22) and (24), we derive the Lyapunov drift

$$\begin{aligned}\Delta(t) =& \mathbb{E}\left[\mathscr{L}(t+1) - \mathscr{L}(t)|\tilde{Q}^t\right]\\ =& \frac{1}{2}\sum_{n=1}^{N}\mathbb{E}\left[(\tilde{Q}_n^{t+1})^2 - (\tilde{Q}_n^t)^2|\tilde{Q}^t\right]\\ =& \frac{1}{2}\sum_{n=1}^{N}\mathbb{E}\left[(\tilde{Q}_n^t + e_n^t - E_n^t)^2 - (\tilde{Q}_n^t)^2|\tilde{Q}^t\right]\\ \leq& \frac{1}{2}\sum_{n=1}^{N}\mathbb{E}\left[(e_n^t)^2 + (E_n^t)^2 + 2\tilde{Q}_n^t(e_n^t - E_n^t)|\tilde{Q}^t\right].\end{aligned} \tag{25}$$

Recall that the upper bound of e_n^t is $E_H^{\max}$, and the upper bound of E_n^t is $E^{\max}$. Setting $\Phi = \frac{1}{2}\sum_{n=1}^{N}\left[(E_H^{\max})^2 + (E^{\max})^2\right]$, it is obvious that

$$\Delta(t) \leq \Phi + \sum_{n=1}^{N}\tilde{Q}_n^t\mathbb{E}\left[e_n^t - E_n^t|\tilde{Q}^t\right]. \quad \square \tag{26}$$

By extending the Lyapunov drift to an optimization problem, our objective is to minimize an upper bound on the following drift-plus-penalty expression in each time slot

$$\begin{aligned}\Delta_V(t) =& \Delta(t) + V\cdot\mathbb{E}\left[cost^t|\tilde{Q}^t\right]\\ \leq& \Phi + \sum_{n=1}^{N}\mathbb{E}\left[\tilde{Q}_n^t(e_n^t - E_n^t) + V\cdot cost_n^t|\tilde{Q}^t\right].\end{aligned} \tag{27}$$

In Eq. (27), Φ is a constant, so $\mathbb{P}2$ is further represented as a minimization of $\sum_{n=1}^{N}\mathbb{E}\left[\tilde{Q}_n^t(e_n^t - E_n^t) + V\cdot cost_n^t|\tilde{Q}^t\right]$, *i.e.*, problem $\mathbb{P}3$.

$$\mathbb{P}3: \min_{x_n^t, f_n^t, p_n^t, e_n^t}\sum_{n=1}^{N}\left[\tilde{Q}_n^t(e_n^t - E_n^t) + V\cdot cost_n^t\right]$$

s.t. (18a)–(18c), (18e)–(18i), (19a)

We can solve the original problem by minimizing $\mathbb{P}3$ per time slot. Before proceeding with its solution, we present the following theorem to show that the solution obtained by our algorithm approximates the theoretical solution of the problem $\mathbb{P}2$.

Theorem 2. *The long-term average execution delay and long-term net collected energy have upper bounds, i.e.,*

$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\sum_{n=1}^{N}\mathbb{E}\left[cost_n^{t,*}\right]\le\frac{\Phi}{V}+cost^{\text{opt}}, \tag{29}$$

$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\sum_{n=1}^{N}\mathbb{E}\left[e_n^t-E_n^{t,*}\right]\le\Phi+V(cost^{\max}-cost^{\text{opt}}), \tag{30}$$

where $*$ indicates the high-quality solution found by our algorithm, and $cost_n^{t,*}$ is the corresponding execution cost of the solution. $cost^{\max}=Ncost_n^{\max}$ is the system's maximum execution cost. $cost^{\text{opt}}$ is the cost corresponding to the theoretically optimal solution to the problem $\mathbb{P}2$.

Proof. Due to page limit, the proof of Theorem 2 is provided in Appendix A. □

Theorem 2 shows that the bound of the cost performance of problem $\mathbb{P}3$ is close to that of problem $\mathbb{P}2$. Next, we divide the problem $\mathbb{P}3$ into two parts for optimization, the energy harvesting part $\tilde{Q}_n^t e_n^t$ and the execution costing part $-\tilde{Q}_n^t E_n^t+V\cdot cost_n^t$. We design the optimal energy harvesting and the optimal task offloading and resource allocation algorithms to handle these two parts, respectively.

4.3. Optimal energy harvesting

The optimal harvested energy of D_n can be determined by solving the following linear programming problem

$$\mathbb{P}_{EH}:\min_{e_n^t}\tilde{Q}_n^t e_n^t$$

s.t. (18e)

Let e_n^{t*} denote the optimal solution of $\mathbb{P}_{EH}$. It is obvious that when $\tilde{Q}_n^t\le 0$, *i.e.*, $Q_n^t\le\theta_n$, the optimal strategy is to maximize the harvested energy, *i.e.*, setting $e_n^{t*}=E_{n,H}^t$. This means that D_n should fully harvest the available energy in the slot t. Conversely, when $\tilde{Q}_n^t>0$, *i.e.*, $Q_n^t>\theta_n$, the optimal strategy is to minimize the energy input, resulting in $e_n^{t*}=0$, implying that D_n will not harvest energy in the slot t. Thus, we have

$$e_n^t=\begin{cases}E_{n,H}^t, & \tilde{Q}_n^t\le 0\\ 0, & \tilde{Q}_n^t>0\end{cases}. \tag{31}$$

Further, we can derive the range of values of Q_n^t by the following theorem.

Theorem 3. *According to the optimal energy harvesting strategy, there exists an upper bound on the battery energy of D_n, i.e., $Q_n^t\in\left[0,\theta_n+E_H^{\max}\right]$, $\forall t\in\mathcal{T}$.*

Proof. It is obvious that $Q_n^t\in\left[0,\theta_n+E_H^{\max}\right]$ is true when $t=0$. When $t=T$, we assume that the induction hypothesis holds, *i.e.*, $Q_n^T\in\left[0,\theta_n+E_H^{\max}\right]$.

- If $Q_n^T\le\theta_n$, from Eq. (31), $e_n^T=E_{n,H}^T$ holds. Thus, $Q_n^T+e_n^{T*}\le\theta_n+e_n^{T*}\le\theta_n+E_H^{\max}$. According to Eq. (16), we have $Q_n^{T+1}\le Q_n^T+e_n^{T*}\le\theta_n+E_H^{\max}$.
- If $\theta_n<Q_n^T\le\theta_n+E_H^{\max}$, from Eq. (31), it holds $e_n^{T*}=0$. Then we have $Q_n^{T+1}\le Q_n^T\le\theta_n+E_H^{\max}$.

In conclusion, when $t=T+1$, $Q_n^{T+1}\in\left[0,\theta_n+E_H^{\max}\right]$. We complete the mathematical induction, *i.e.*, $\forall t\in\mathcal{T}$, it holds that $Q_n^t\in\left[0,\theta_n+E_H^{\max}\right]$. □

4.4. Optimal resource allocation

After solving the energy harvesting part of $\mathbb{P}3$, we solve the execution cost part, which can be expressed as the following problem.

$$\mathbb{P}_{TORA}:\min_{x_n^t,f_n^t,p_n^t}\sum_{n=1}^{N}\left[-\tilde{Q}_n^t E_n^t+V\cdot cost_n^t\right]$$

s.t. (18a)–(18c), (18f)–(18i), (19a)

Algorithm 1: Optimal Resource Allocation

Input: Current slot $h_{mn}^t, H_m^t, N_m^t, N^t$, task set Γ, a given fixed task offloading decision X^t

Output: Resource allocation decision

```
for Each ED do
    if Task executing locally then
        Calculate the optimal frequency f_{n,l}^{t*} using Eq. (32);
    else if Task executing at edge then
        Calculate the optimal transmission power p_n^{t*} using Eq. (33);
        Calculate the optimal frequency allocation for edge servers f_{n,e}^{t*} using Eq. (34);
    else if Task executing at cloud then
        Calculate the optimal transmission power p_n^{t*} using Eq. (33);
    else if Dropped or no-generated tasks then
        Continue;
return Resource allocation decision;
```

It should be noted that once the task offloading strategy is determined, $\mathbb{P}_{TORA}$ can be reframed as a resource allocation problem $\mathbb{P}_{RA}$. The resource allocation for tasks executed at the local, edge, and cloud levels, as well as for dropped tasks, is independent of each other. In other words, these four cases are fully decoupled, which enables us to decompose $\mathbb{P}_{RA}$ into four independent subproblems. Specifically, when task τ_n^t is dropped, the objective of $\mathbb{P}_{RA}$ simplifies to a constant $V\lambda$. Therefore, we will only discuss the other three cases in the following. The proposed resource allocation strategy is formally described in **Algorithm** 1.

4.4.1. Local execution

For each task τ_n^t executed locally on D_n, we aim to obtain the optimal CPU frequency for D_n, which is expressed as

$$\mathbb{P}_{RA,l}:\min_{f_{n,l}^t}-\tilde{Q}_n^t k\left(f_{n,l}^t\right)^2+V\cdot\frac{C_n^t}{f_{n,l}^t}$$

s.t. (18a), (18f), (19a)

We first derive the feasible range of $\mathbb{P}_{RA,l}$. From Eqs. (14) and (18f), $L_n^t=\frac{C_n^t}{f_{n,l}^t}\le Y$, thus the local device's frequency satisfies $f_{n,l}^t\ge\frac{C_n^t}{Y}$. Additionally, from Eqs. (18a) and (19a), we have $f_{n,l}^t\le f_l^{\max}$ and $\sqrt{\frac{E^{\min}}{kC_n^t}}\le f_{n,l}^t\le\sqrt{\frac{E^{\max}}{kC_n^t}}$, respectively. Therefore, we can derive the lower and upper bounds of $f_{n,l}^t$ as $f_{n,L}^t=\max\left\{\sqrt{\frac{E^{\min}}{kC_n^t}},\frac{C_n^t}{Y}\right\}$ and $f_{n,U}^t=\min\left\{\sqrt{\frac{E^{\max}}{kC_n^t}},f_l^{\max}\right\}$, respectively.

Let the objective function of $\mathbb{P}_{RA,l}$ be $F(f_{n,l}^t)=-\tilde{Q}_n^t k\left(f_{n,l}^t\right)^2+V\cdot\frac{C_n^t}{f_{n,l}^t}$. By taking the derivative of $F(f_{n,l}^t)$, we have $F'(f_{n,l}^t)=-2\tilde{Q}_n^t k f_{n,l}^t C_n^t-\frac{VC_n^t}{f_{n,l}^{t\,2}}$. When $\tilde{Q}_n^t\ge 0$, $F'(f_{n,l}^t)<0$ holds. Therefore, $F(f_{n,l}^t)$ is monotonically decreasing, and the optimal value of $f_{n,l}^t$ is $f_{n,l}^{t*}=f_{n,U}^t$. When $\tilde{Q}_n^t<0$, we set $F'(f_{n,l}^t)=0$ and find the extremal point $\hat{f}_{n,l}^t=\left(\frac{-V}{2\tilde{Q}_n^t k}\right)^{\frac{1}{3}}$. The value of $\hat{f}_{n,l}^t$ is discussed as follows.

- If $\hat{f}_{n,l}^t\le f_{n,L}^t$, *i.e.*, $\tilde{Q}_n^t\le\frac{-V}{2k(f_{n,L}^t)^3}$, then $F(f_{n,l}^t)$ is monotonically increasing, and we have $f_{n,l}^{t*}=f_{n,L}^t$.
- If $f_{n,L}^t<\hat{f}_{n,l}^t<f_{n,U}^t$, *i.e.*, $\frac{-V}{2k(f_{n,L}^t)^3}<\tilde{Q}_n^t<\frac{-V}{2k(f_{n,U}^t)^3}$, then $F(f_{n,l}^t)$ reaches the minimal value at $\hat{f}_{n,l}^t$, and $f_{n,l}^{t*}=\hat{f}_{n,l}^t$.
- If $\hat{f}_{n,l}^t\ge f_{n,U}^t$, *i.e.*, $\frac{-V}{2k(f_{n,U}^t)^3}\le\tilde{Q}_n^t<0$, then $F(f_{n,l}^t)$ is monotonically decreasing, and we have $f_{n,l}^{t*}=f_{n,U}^t$.

In conclusion, the optimal allocation of local frequency is

$$f_{n,l}^{t*} = \begin{cases} f_{n,L}^t, & \tilde{Q}_n^t \le \frac{-V}{2k(f_{n,L}^t)^3} \\ \hat{f}_{n,l}^t, & \frac{-V}{2k(f_{n,L}^t)^3} < \tilde{Q}_n^t < \frac{-V}{2k(f_{n,U}^t)^3} \\ f_{n,U}^t, & \tilde{Q}_n^t \ge \frac{-V}{2k(f_{n,U}^t)^3} \end{cases}. \tag{32}$$

4.4.2. Edge server execution

Let $\hat{D}_m^t$ represent the set of EDs offloading tasks to the edge server S_m at slot t. For each edge server S_m, the resource allocation problem can be represented as

$$\mathbb{P}_{RA,e}: \min_{f_{n,e}^t, p_n^t} \sum_{D_n \in \hat{D}_m^t} -\tilde{Q}_n^t \frac{p_n^t A_n^t}{r_{n,u}^t} + V\left(\frac{A_n^t}{r_{n,u}^t} + \frac{C_n^t}{f_{n,e}^t} + \frac{U_n^t}{r_{n,d}^t}\right)$$

s.t. (18b), (18c), (18f), (19a)

The objective of $\mathbb{P}_{RA,e}$ can be expressed as the sum of three terms: $-\tilde{Q}_n^t \frac{p_n^t A_n^t}{r_{n,u}^t} + V\frac{A_n^t}{r_{n,u}^t}$, $V\frac{C_n^t}{f_{n,e}^t}$, and $V\frac{U_n^t}{r_{n,d}^t}$. Notably, the third term is a constant. To optimize the first two terms, we determine the optimal transmission power and the optimal frequency assignment separately.

Optimal Transmission Power. If task τ_n^t is offloaded, it is transmitted to the edge server over the wireless channel. The optimal transmission power allocation problem for D_n is expressed as

$$\mathbb{P}_{RA,ep}: \min_{p_n^t} -\tilde{Q}_n^t \frac{p_n^t A_n^t}{r_{n,u}^t} + V \cdot \frac{A_n^t}{r_{n,u}^t}$$

s.t. (18c), (18f), (19a)

We first derive the range of $p_n^t {r_{n,u}^t}^{-1}$ by the following lemma.

Lemma 1. *The function $H(p_n^t) = p_n^t {r_{n,u}^t}^{-1}$ is monotonically increasing with respect to p_n^t, and it takes values in the range $\left(\sigma^2 \xi {A_n^t}^{-1} \ln 2, +\infty\right)$, where $\xi = A_n^t N_m^t \left(B_{m,u} h_{mn}^t\right)^{-1}$.*

Proof. Due to page limit, the detailed proof is provided in Appendix B.[1] □

Based on Lemma 1, we proceed to derive the optimal transmission power p_n^{t*}. As a preliminary step, we first derive the feasible range of $\mathbb{P}_{RA,ep}$. From Eq. (18f), $L_n^t = A_n^t {r_{n,u}^t}^{-1} \le Y$ implies that the transmission power satisfies $p_n^t \ge \frac{\sigma^2}{h_{mn}^t}(2^{\xi h_{mn}^t Y^{-1}} - 1)$. From Eq. (18c), we also have $p_n^t \le p_l^{\max}$. Additionally, from Eq. (19a), the inequality $E^{\min} \le p_n^t A_n^t {r_{n,u}^t}^{-1} \le E^{\max}$ holds. Solving this inequality yields the bounds $p_{n,E^{\min}}^t \le p_n^t \le p_{n,E^{\max}}^t$. Note that $p_{n,E^{\min}}^t$ and $p_{n,E^{\max}}^t$ cannot be expressed in closed form. In conclusion, we derive the lower and upper bounds of p_n^t as

$$p_{n,L}^t = \begin{cases} \max\{p_{n,Y}^t, p_{n,E^{\min}}^t\}, & \xi\sigma^2 \ln 2 < E^{\min} \\ p_{n,Y}^t, & \xi\sigma^2 \ln 2 \ge E^{\min} \end{cases},$$

and

$$p_{n,U}^t = \begin{cases} \min\{p_l^{\max}, p_{n,E^{\max}}^t\}, & \xi\sigma^2 \ln 2 < E^{\max} \\ 0, & \xi\sigma^2 \ln 2 \ge E^{\max} \end{cases},$$

respectively, where $p_{n,Y}^t = \frac{\sigma^2}{h_{mn}^t}(2^{\xi h_{mn}^t Y^{-1}} - 1)$.

For ease of presentation, the first term of the objective of $\mathbb{P}_{RA,e}$ is denoted as $F(p_n^t) = -\tilde{Q}_n^t \frac{p_n^t A_n^t}{r_{n,u}^t} + V \cdot \frac{A_n^t}{r_{n,u}^t}$. By taking the derivative of $F(p_n^t)$, we have

$$F'(p_n^t) = \frac{\xi h_{mn}^t}{\log_2^2 \varepsilon} \cdot G(p_n^t),$$

where $G(p_n^t) = -\tilde{Q}_n^t \log_2 \varepsilon + (\tilde{Q}_n^t p_n^t - V)\frac{h_{mn}^t}{\sigma^2 \varepsilon \ln 2}$ and $\varepsilon = 1 + h_{mn}^t p_n^t \sigma^{-2}$.

[1] The range of function $H(p_n^t)$ is derived from [15]. Since the proof is not provided in [15], we prove it in the paper.

When $\tilde{Q}_n^t \ge 0$, it holds $F'(p_n^t) < 0$. Therefore, $F(p_n^t)$ is monotonically decreasing, and the optimal value of p_n^t is $p_n^{t*} = p_{n,U}^t$. When $\tilde{Q}_n^t < 0$, the derivative of $G(p_n^t)$ indicates that $G(p_n^t)$ is monotonically increasing. Additionally, $G(0) = \frac{-V h_{mn}^t}{\sigma^2 \ln 2}$ and $\lim_{p_n^t \to +\infty} G(p_n^t) = +\infty$. Therefore, there exists $\hat{p}_n^t \in (0, +\infty)$ making $F'(p_n^t) = 0$. The value of $\hat{p}_n^t$ is discussed as follows.

- If $\hat{p}_n^t \le p_{n,L}^t$, then $F(p_n^t)$ is monotonically increasing, and we have $p_n^{t*} = p_{n,L}^t$.
- If $p_{n,L}^t < \hat{p}_n^t < f_{n,U}^t$, then $F(p_n^t)$ reaches a minimal value at $\hat{p}_n^t$, and $p_n^{t*} = \hat{p}_n^t$.
- If $\hat{p}_n^t \ge p_{n,U}^t$, then $F(p_p^t)$ is monotonically decreasing, and we have $p_n^{t*} = p_{n,U}^t$.

In conclusion, the optimal transmission power of D_n is

$$p_n^{t*} = \begin{cases} p_{n,U}^t, & \tilde{Q}_n^t \ge 0 \text{ or } \tilde{Q}_n^t < 0, p_n^{t'} \ge p_{n,U}^t \\ p_n^{t'}, & \tilde{Q}_n^t < 0, p_{n,L}^t < p_n^{t'} < p_{n,U}^t \\ p_{n,L}^t, & \tilde{Q}_n^t < 0, p_n^{t'} \le p_{n,L}^t \end{cases}. \tag{33}$$

Optimal Frequency Allocation. For an edge server S_m, multiple offloaded tasks compete for computation resources within a given time slot. The optimal frequency allocation problem for S_m is expressed as

$$\mathbb{P}_{RA,ef}: \min_{f_{n,e}^t} \sum_{D_n \in \hat{D}_m^t} V \cdot \frac{C_n^t}{f_{n,e}^t}$$

s.t. (18b), (18f), (19a)

According to Eq. (18b), the computation resources allocated to each task τ_n^t by the edge server at slot t are interdependent, making it infeasible to determine the optimal allocated frequency for each task independently. To address this issue, we employ the Lagrange Multiplier method that transforms a complex optimization problem with multiple constraints into an unconstrained one [46]. First, we represent the resource constraint of the edge server as $H(f_{n,e}^t) = f_{m,e}^{\max} - \sum_{D_n \in \hat{D}_m^t} f_{n,e}^t$. In general, a higher CPU frequency assigned to a task leads to lower task delay and better user experience. Therefore, we assume $H(f_{n,e}^t) = 0$ which indicates that the computing capacity of the edge server is fully utilized. Based on this, we construct the Lagrangian function $\mathcal{L}(f_{n,e}^t, \eta) = \sum_{D_n \in \hat{D}_m^t} V \cdot \frac{C_n^t}{f_{n,e}^t} + \eta H(f_{n,e}^t)$. By taking the total derivatives of $\mathcal{L}(f_{n,e}^t, \eta)$, we have $\mathcal{L}'_{f_{n,e}^t} = -\frac{VC_n^t}{(f_{n,e}^t)^2} - \eta$ and $\mathcal{L}'_\eta = f_{m,e}^{\max} - \sum_{D_n \in \hat{D}_m^t} f_{n,e}^t$. When $\mathcal{L}'_{f_{n,e}^t} = \mathcal{L}'_\eta = 0$, $f_{n,e}^t = \sqrt{\frac{VC_n^t}{-\eta}}$ and $\sum_{D_n \in \hat{D}_m^t} f_{n,e}^t = f_{m,e}^{\max}$ hold. From this, we can obtain $\sum_{D_n \in \hat{D}_m^t} \sqrt{\frac{VC_n^t}{-\eta}} = f_{m,e}^{\max}$ and $\sqrt{-\eta} = \frac{\sum_{D_n \in \hat{D}_m^t} \sqrt{VC_n^t}}{f_{m,e}^{\max}}$. Thus, the optimal frequency of task τ_n^t can be derived as

$$f_{n,e}^{t*} = \frac{f_{m,e}^{\max} \sqrt{c_n^t}}{\sum_{D_n \in \hat{D}_m^t} \sqrt{c_n^t}}. \tag{34}$$

4.4.3. Cloud server execution

The cloud server typically has significantly greater computation power compared to EDs and edge servers. Following [41], we assume that each task offloaded to the cloud server is executed with a constant CPU frequency. As a result, for each task τ_n^t offloaded to the cloud server, the optimization problem focuses on determining the optimal transmission power for D_n.

$$\mathbb{P}_{RA,cp}: \min_{p_n^t} -\tilde{Q}_n^t \frac{p_n^t A_n^t}{r_{n,u}^t} + V \cdot \frac{A_n^t}{r_{n,u}^t}$$

s.t. (18c), (18f), (19a)

Similar to the case of task offloading execution on edge servers, we obtain the optimal transmission power p_n^{t*} for task offloading execution on the cloud server by solving the optimization problem $\mathbb{P}_{RA,cp}$. For

brevity, the detailed derivation is omitted here. The p_n^{t*} is formulated as

$$p_n^{t*} = \begin{cases} p_{n,U}^t, & \tilde{Q}_n^t \geq 0 \text{ or } \tilde{Q}_n^t < 0, p_n^{t'} \geq p_{n,U}^t \\ p_n^{t'}, & \tilde{Q}_n^t < 0, p_{n,L}^t < p_n^{t'} < p_{n,U}^t \\ p_{n,L}^t, & \tilde{Q}_n^t < 0, p_n^{t'} \leq p_{n,L}^t \end{cases}. \tag{35}$$

4.5. MDPSO-based task offloading algorithm

After solving the resource allocation problem, the task offloading problem is expressed as

$$\mathbb{P}_{TO}: \min_{x_n^t} \tilde{Q}_n^t(e_n^t - E_n^t) + V \cdot cost_n^t$$

s.t. (18g)–(18i)

Discrete Particle Swarm Optimization (DPSO) algorithm [47] is commonly used to obtain discrete task offloading decisions. However, the inherent randomness in population initialization may cause DPSO to converge local optima. To overcome this shortage, we propose a task offloading scheme based on the **M**ultiple **D**iscrete **P**article **S**warm **O**ptimization (MDPSO). The MDPSO algorithm divides the population of size S into γ smaller sub-populations, each containing S/γ particles. Each sub-population explores offloading decisions in parallel, which promotes diversity in the search process. Furthermore, the evolution of sub-populations can be distributed, accelerating the overall algorithmic performance. When the offloading decisions of all the sub-populations converge, crossover and exchange operations are applied to the high-quality offloading decisions obtained from the individual sub-populations to further improve the solution quality. After that, the high-quality solution is selected as the final task offloading decision output by the MDPSO algorithm. Fig. 3 gives an example to show the different search processes of DPSO and MDPSO.

In the MDPSO algorithm, the particle's position information $X^t = [x_1^t, \ldots, x_n^t, \ldots, x_N^t]$ represents the task offloading decision at slot t. Specifically, the nth dimension $x_n^t = [x_{n,l}^t, x_{n,e}^t, x_{n,c}^t, x_{n,d}^t]^\top$ of X^t denotes the offloading decision for task τ_n^t. Each element of x_n^t must satisfy the constraints outlined in Eq. (18h). The particle's velocity $Y^t = [y_1^t, \ldots, y_n^t, \ldots, y_N^t]$ represents the tendency of the task to be offloaded to different execution locations, where each y_n^t is a 4×1 vector. The velocity update rule is given by

$$Y^{t,k+1} = \omega Y^{t,k} + c_1 r_1 \left(X_b^{t,k} - X^{t,k}\right) + c_2 r_2 \left(X_{gb}^{t,k} - X^{t,k}\right), \tag{36}$$

where $X_b^{t,k}$ denotes the better offloading decisions found by particles after the kth iteration, and $X_{gb}^{t,k}$ denotes the high-quality offloading decisions found by the sub-populations. ω is the inertia weight, c_1, c_2 are the learning factors, and r_1, r_2 are random numbers. The particle position update rule is

$$X^{t,k+1} = X^{t,k} + Y^{t,k+1}. \tag{37}$$

To evaluate the quality of offloading decisions, we set the objective of problem $\mathbb{P}3$ as the particle's fitness, *i.e.*,

$$fitness^{t,k} = \left[\tilde{Q}_n^t(e_n^t - E_n^t) + V \cdot cost_n^t\right]^k. \tag{38}$$

The proposed MDPSO-based task offloading scheme is detailed in **Algorithm** 2. The algorithm begins by dividing the population into sub-populations to enhance diversity during the evolution process (Line 1). Subsequently, we initialize the offloading decisions and the corresponding optimal resource allocations and fitness (Lines 2–4). The algorithm iterates over all particles to optimize offloading decisions, terminating when the decisions within each sub-population converge (Lines 5–16). During each iteration, the velocity and position of particles representing offloading decisions are updated according to Eqs. (36) and (37) (Line 7). Then **Algorithm** 1 is called to decide the optimal resource allocation of $X_j^{t,k}$ (Line 8). To ensure the reliability of task execution, a critical constraint is imposed: if the reliability associated with a computed decision does not meet the predefined threshold (Line 11), the fitness value for that decision is set to positive infinity, excluding it from further consideration (Line 12). We employ stochastic cross-swapping [48] to enable the exchange of high-quality decisions between sub-populations, facilitating convergence toward a global optimum by leveraging the strengths of diverse local high-quality (Line 18–22). Finally, the algorithm outputs the high-quality task offloading decision X^{t*} (Line 23).

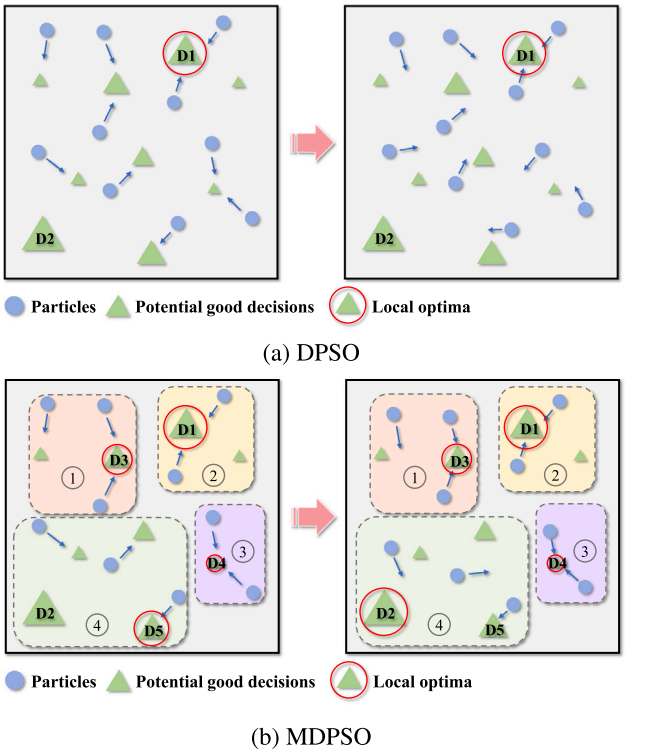


Fig. 3. Diagram of standard DPSO and MDPSO. Blue circles denote particles (offloading decisions), potential good decisions represent offloading decisions with better fitness, and local optima mark the current high-quality offloading decision. In DPSO, particles converge to a global high-quality decision (*e.g.*, D1), potentially missing better solutions like D2. MDPSO splits particles into sub-populations that independently explore the search space, discovering multiple high-quality decisions (*e.g.*, D1, D2, D3, D4, D5). MDPSO reduces the risk of local optima and achieves superior offloading decisions. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

4.6. The overall DTORA framework

The overall DTORA framework is summarized in **Algorithm** 3. First, the framework uses Lyapunov optimization techniques to decouple the original problem. (Lines 1–2). Next, it obtains the task offloading and resource allocation strategy for each time slot (Lines 3–8). Specifically, the framework determines the optimal energy harvesting strategy (Line 6) and then calls **Algorithm** 2 to derive the high-quality task offloading decision (Line 7). In **Algorithm** 2, **Algorithm** 1 is called to derive the optimal resource allocation solution. At the end of each slot, the battery power of all EDs is updated for the decision-making of the next slot (Line 8).

The time complexity of DTORA is analyzed as follows. Lines 1–2 are executed once with a constant complexity $O(1)$. Line 4 iterates over N EDs, resulting in a complexity of $O(|N|)$. For lines 5–6, the time complexity is at most $O(|N|)$. Line 7 calls **Algorithm** 2 to solve the task offloading problem with a complexity dominated by the configuration combinations. In intra-group iteration, each of γ groups processes $\frac{S}{\gamma}$ particles. The velocity and position updates for each group take $\frac{S}{\gamma} \cdot N \cdot \rho$ and the fitness evaluation for $N \cdot \rho$ tasks requires $O(1)$ time per task,

Algorithm 2: MDPSO based Task Offloading

Input: Current slot environment attributes, task generation situation
Output: Task offloading decision

Generate a population of size S and homogenize the population into γ sub-populations;
Initialize all offloading decisions $X_1^{t,0}, X_2^{t,0}, \ldots, X_S^{t,0}$;
Call **Algorithm 1** to calculate the optimal resource allocation;
Update the cost fitness $fitness^{t,k}$ and the high-quality cost fitness $fitness_b^{t,k}$ according to Eq. (38);
while $|fitness_b^{t,k} - fitness_b^{t,k-1}| < \varphi$ **do**
  for Each particle j **do**
    Update $Y_j^{t,k}$ and $X_j^{t,k}$ according to Eq. (36) and Eq. (37);
    Call **Algorithm 1** to decide the optimal resource allocation of $X_j^{t,k}$;
    Calculate the task execution reliability $R_j^{t,k}$ according to Eq. (11);
    Calculate $fitness_j^{t,k}$ according to Eq. (38);
    if $R_j^k < R^{\text{th}}$ **then**
      $fitness_j^{t,k} = +\infty$;
    else if $fitness_j^{t,k} < fitness_{j,b}^{t,k}$ **then**
      $fitness_{j,b}^{t,k} = fitness_j^{t,k}$;
      $X_{i,b}^{t,k} = X_j^{t,k}$;
  Update the global high-quality cost fitness $fitness_{i,g}^{t,k}$ of each sub-population i;
Obtain X^{t*} with the minimum fitness of decision;
while $\exists X_{i_1,b}^{t,k} \neq X_{i_2,b}^{t,k}$ **do**
  $X_{\text{new}}^{t,k} = Crossswap(X_{i_1,b}^{t,k}, X_{i_2,b}^{t,k})$;
  if $fitness_{\text{new}}^{t,k} < fitness^{t*}$ **then**
    $fitness^{t*} = fitness_{\text{new}}^{t,k}$;
    $X^{t*} = X_{\text{new}}^{t,k}$;
return Task offloading decision X^{t*};

Algorithm 3: DTORA Framework

Set the virtual energy queue based on Eqs. (21) and (22);
Obtain the decoupled problem $\mathbb{P}3$ using Eq. (27);
for each slot **do**
  Get the system state of the current time slot t and the generated task set Γ;
  for Each ED D_n **do**
    Calculate the optimal energy harvesting e_n^{t*} according to Eq. (31);
  Call **Algorithm 2** to derive the task offloading decision X^{t*};
  Update the battery energy of all EDs according to Eq. (16);

yielding $\frac{S}{\gamma} \cdot N \cdot \rho$. Over T iterations, the total complexity is at most $T \cdot 2\gamma \cdot \frac{S}{\gamma} \cdot N \cdot \rho = O(T \cdot S \cdot N \cdot \rho)$. For inter-group crossover, exchanging γ high-quality solutions has complexity $O(\gamma \cdot N \cdot \rho)$. Therefore, the overall complexity of MDPSO is $O(|\Theta|) = O(T \cdot S \cdot N \cdot \rho)$ since $T \cdot S \gg \gamma$, where $|\Theta| = T \cdot S \cdot N \cdot \rho$. Here, T, S, N, and ρ are the number of iterations, particles, EDs, and the task generation probability, respectively. Thus, Θ represents the configuration combinations of MDPSO. For line 8, the time complexity is at most $O(|N|)$. Therefore, for **Algorithm 3**, the time complexity is at most $O(1)+O(|N|)+O(|N|)+O(T \cdot M \cdot N \cdot \rho)+O(|N|) = O(|\Theta|)$.

Table 3
Experimental parameters setting.

Variants	Value	Variants	Value
M	5	$r_{m,U}$	100 Mbps
λ	2 ms	$r_{m,D}$	300 Mbps
Y	2 ms	h_{mn}^t	$g_0(d_0/d)^4$
$E_H^{\max}$	0.048 mJ	g_0	−40 dB
$E^{\max}$	2 mJ	d_0	1 m
$E^{\min}$	0.02 mJ	d	60 m–100 m
A_n^t	1000 bits	σ^2	10^{-13} W
U_n^t	200 bits	$f_l^{\max}$	1.5 GHz
1 bits-cycles	600–800	$p_l^{\max}$	0.5 w
W_n^t	(0,1]	k	10^{-28}
μ_0	10^{-2}	$f_{m,e}^{\max}$	3 GHz
δ	3	$f_{n,c}^t$	3 GHz
$B_{n,u}$	35 MHz	P_H	12 mW
$B_{n,d}$	100 MHz		

5. Evaluation

5.1. Experimental settings

The experimental settings are detailed in Table 3.

(1) Scenario: We conduct an EEC system consisting of 30 EDs, 5 edge servers ($M = 5$), and a cloud server. The system is simulated over 3000 time slots, each with a duration of $Y = 2$ ms. The same value $\lambda = 2$ ms is used for the task drop penalty [15]. The energy collected by ED in each slot is uniformly distributed between 0 and $E_H^{\max}$, where $E_H^{\max} = P_H \cdot 2Y$ and the average energy collection power is $P_H = 12$mW [20]. The maximum/minimum output energy of the battery in a slot is $E^{\max} = 2$ mJ/$E^{\min} = 0.02$ mJ [20]. We evaluate the system under five task generation probabilities, *i.e.*, 0.3, 0.5, 0.7, 0.9, and 1. Each task τ_n^t has a data volume $A_n^t = 1000$ bits [49], and returns $U_n^t = 200$ bits result data upon completion. It requires 600–800 CPU cycles to process one bit of data. The fragility factor W_n^t of task τ_n^t is uniformly distributed in the range of (0,1] [14]. The initial failure rate is $\mu_0 = 10^{-2}$ [42] and the sensitivity constant is $\delta = 3$ [14].

(2) Communication: We adopt three resource configurations: Low, Medium, and High, which are shown in Table 4. In the absence of specific instructions, the upload/downlink bandwidth $B_{n,u}/B_{n,d}$ between the EDs and the edge servers are set to 35 MHz and 100 MHz, respectively. The upload/download wired rates $r_{m,U}/r_{m,D}$ between edge and cloud are set to 100 Mbps and 300 Mbps, respectively. The channel power gain h_{mn}^t follows an exponential distribution with a mean value of $g_0(d_0/d)^4$, where $g_0 = -40$ dB, $d_0 = 1$m is the reference distance, and d is randomly distributed between [60,100]m. The noise power on the receiver side is $\sigma^2 = 10^{-13}$ W [41]. The maximum CPU frequency, the maximum transmit power, and the chip relevance factor of each ED are set as $f_l^{\max} = 1.5$ GHz [20], $p_l^{\max} = 0.5$ W, and $k = 10^{-28}$ [15], respectively. The maximum CPU frequencies of the edge servers and the allocated CPU frequency of the cloud server are $f_{m,e}^{\max} = 3$ GHz and $f_{n,c}^t = 3$ GHz, respectively.

5.2. Benchmarks and performance metrics

We compare the proposed DTORA algorithm with the following benchmarks.

- **End Only (End)**: All tasks are executed entirely on the local device without any offloading.
- **Edge Only (Edge)**: All tasks are offloaded to an edge server for execution.
- **Cloud Only (Cloud)**: All tasks are offloaded to a remote cloud server for execution.
- **Random Offloading (Random)**: Randomly generates task offloading decisions and applies our proposed optimal energy harvesting and resource allocation algorithms to optimize the cost.

Table 4
Resource configuration.

Configuration	End-Edge upload bandwidth	End-edge download bandwidth	Edge-cloud upload rate	Edge-cloud download rate	Edge server CPU frequency	Cloud server allocation frequency
Low	25 MHz	80 MHz	80 Mbps	200 Mpbs	2.5 GHz	2.5 GHz
Medium	35 MHz	100 MHz	100 Mbps	300 Mbps	3.0 GHz	3.0 GHz
High	45 MHz	120 MHz	120 Mbps	400 Mbps	3.5 GHz	3.5 GHz

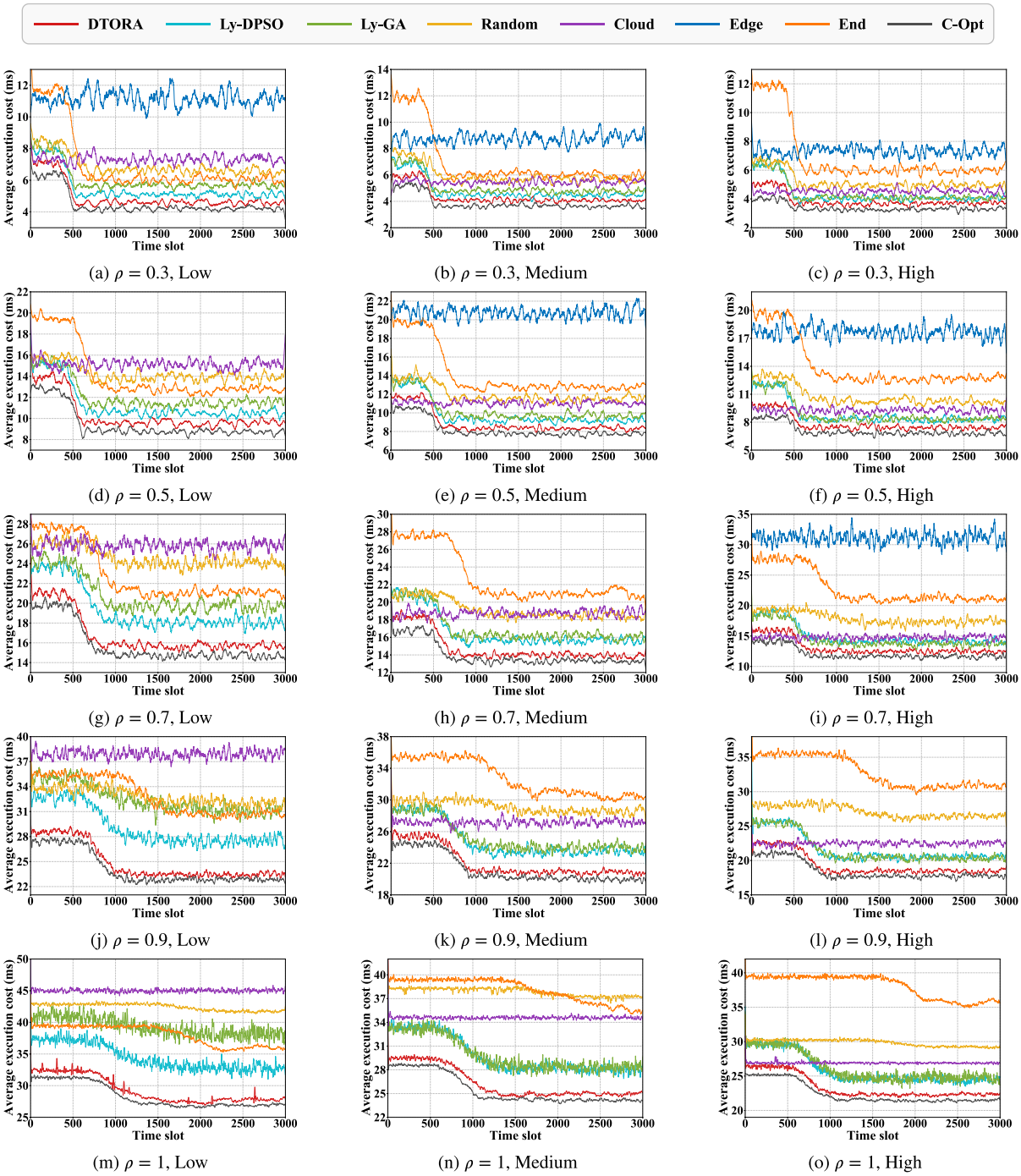
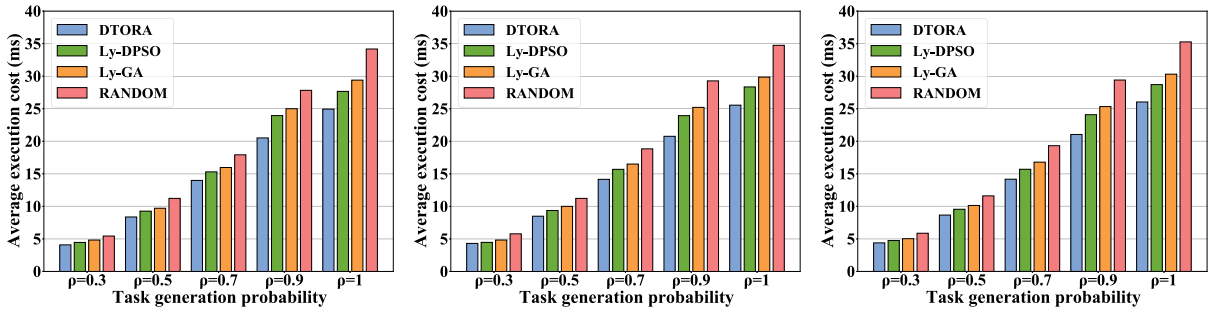


Fig. 4. Average execution costs of different algorithms under different ρ and resource configurations.



(a) $R^{\text{th}} = 0.85$ (b) $R^{\text{th}} = 0.90$ (c) $R^{\text{th}} = 0.95$

Fig. 5. Average execution costs of different algorithms under different ρ and execution reliability.

- **Cost Optimal (C-Opt)**: This method corresponds to the optimal policy derived in Theorem 2, formulating $\mathbb{P}2$ as a static problem focused on minimizing system cost. It employs a solver to compute the decision without long-term energy constraints, providing performance bounds for the problem.
- **Lyapunov based Genetic Algorithm (Ly-GA)** [50]: This method uses Lyapunov optimization to decouple $\mathbb{P}1$ to $\mathbb{P}3$, with GA determining task offloading. In the GA algorithm, our proposed optimal energy harvesting and resource allocation algorithms are applied to calculate the fitness of the task offloading decision.
- **Lyapunov based Discrete Particle Swarm Optimization (Ly-DPSO)**: Similarly to Ly-GA, this method employs DPSO algorithm to obtain the task offloading decision. In the DPSO algorithm, our proposed optimal energy harvesting and resource allocation algorithms are applied to calculate the fitness of the task offloading decision.

To assess the performance of our solutions, we utilize the following metrics in our numerical evaluation.

- **Average Execution Cost**: After the EDs execute the offloading decision, we can calculate the average execution cost under different algorithms.
- **Execution Reliability**: We use the numerical values calculated by Eq. (12) to assess the execution reliability of the tasks.
- **Battery Energy**: We measure the battery energy of each ED in each slot to assess the effectiveness of DTORA in stabilizing battery energy.

5.3. Experimental results

In this section, we first evaluate the impact of the DTORA algorithm on the average execution cost, reliability, and battery energy. Then we assess the influence of various parameters on our algorithm, *e.g.*, the perturbation parameter θ, the weight V, and the lower bound of the battery output energy $E_{\min}$. Without loss of generality, we set the number of sub-populations $\gamma = 3$ for the subsequent evaluation.

5.3.1. Algorithm comparison

Fig. 4 compares the average execution cost of different algorithms. As shown in the figure, the algorithms that involve local execution, *i.e.*, DTORA, Ly-DPSO, Ly-GA, Random, End, and C-Opt, often incur higher initial cost, but tend to stabilize at lower levels over time. For example, in Fig. 4(b), the cost of the DTORA algorithm stays around 6 ms for the first 500 slots and stabilizes around 4 ms after that. This is attributed to the initial insufficient battery energy of the ED, which leads to low CPU frequency during local execution, causing a high execution delay. Additionally, limited battery energy causes frequent task dropping, further elevating the cost. As time progresses, the battery energy gradually accumulates, enabling a higher CPU frequency for local execution and fewer task drops, resulting in lower execution costs.

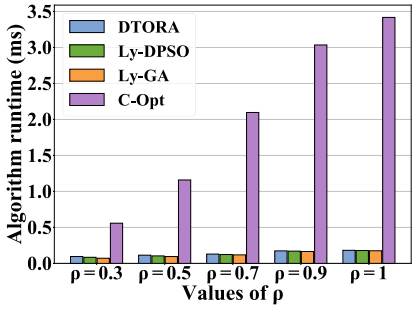


Fig. 6. Comparison of runtime for DTORA, Ly-DPSO, Ly-GA, and C-Opt algorithms.

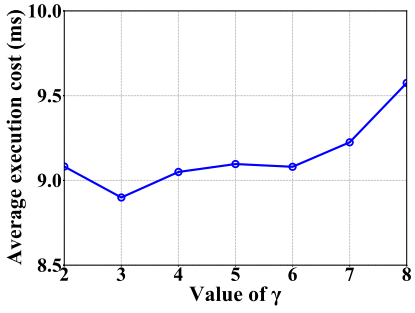
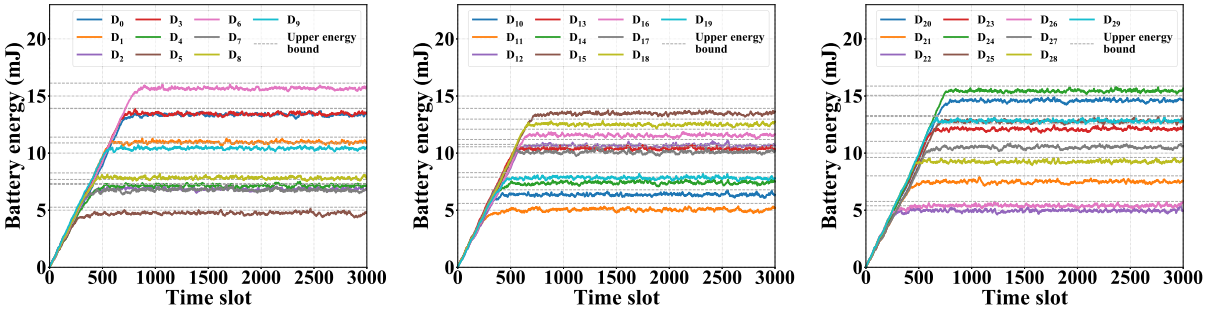


Fig. 7. Performance of DTORA under different γ values.

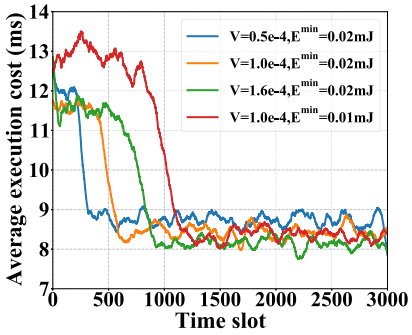
Unlike the above algorithms, the Edge and Cloud algorithms maintain a stable cost value throughout the process. Compared to other algorithms, the C-Opt always achieves the lowest execution cost, as it achieves an optimal solution. In addition, our DTORA algorithm performs close to the C-Opt algorithm and better than the other algorithms in all scenarios. For example, as shown in Fig. 4(j), the stabilization costs for the Cloud, Random, Ly-GA, End, Ly-DPSO, DTORA, and C-Opt algorithms are 37.9 ms, 32.0 ms, 31.2 ms, 30.7 ms, 27.5 ms, 23.4 ms, and 22.9 ms, respectively. Compared to the Cloud, Random, Ly-GA, End, and Ly-DPSO algorithms, DTORA reduces the execution cost by 38.2%, 26.9%, 25.0%, 23.6%, and 14.8%, respectively. Moreover, compared to the C-Opt algorithm, DTORA shows only a 2.5% gap in performance.

Additionally, as the task generation probability ρ increases, the cost of all algorithms increases, and the convergence rate of most algorithms slows down. For example, as shown in Fig. 4(a)(d)(g)(j)(m), the DTORA algorithm converges to lower cost values at approximately 500, 650, 900, 1100, and 1600 time slots for ρ values of 0.3, 0.5, 0.7, 0.9, and 1, respectively. Furthermore, the higher the resource configuration, the lower the execution cost for all algorithms except the End algorithm. For example, in Fig. 4(a)(b)(c), the stabilization cost of the Ly-DPSO algorithm is about 5 ms, 4.5 ms, and 4 ms, respectively. It is observed

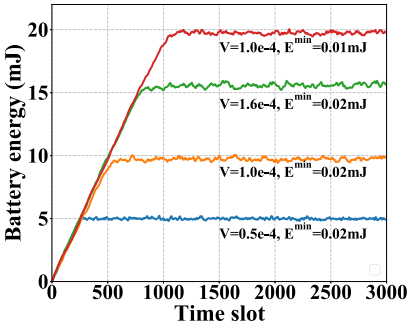


(a) EDs $D_0 - D_9$ (b) EDs $D_{10} - D_{19}$ (c) EDs $D_{20} - D_{29}$

Fig. 8. Battery energy levels of EDs.



(a) Average execution cost under different parameters



(b) Battery energy under different parameters

Fig. 9. Impact of V and $E^{\min}$ on average execution cost and battery energy.

that the Edge algorithm is more sensitive to both ρ and resource allocation compared to the End and Cloud algorithms. For example, in Fig. 4(a)(b)(c), the costs of the Edge and Cloud algorithms under Low, Medium, and High resource configurations are approximately 11 ms, 9 ms, 7 ms and 7 ms, 5.5 ms, 4 ms, respectively. This is because ED D_n executes at most one task under the End strategy in a single slot, whereas the Edge and Cloud algorithms need to handle multiple tasks. Note that due to the limited resources of the edge servers, the cost of the Edge algorithm is too high, and we only display it in Fig. 4(a)(b)(c)(e)(f)(i).

Fig. 5 shows the impact of task execution reliability constraints on the average execution cost. Since the offloading decisions generated by the End, Edge, Cloud, and C-Opt algorithms are deterministic and cannot be adjusted based on reliability, the analysis focuses solely on the DTORA, Ly-DPSO, Ly-GA, and Random algorithms. Among these, the DTORA algorithm consistently achieves the lowest cost. For example, in Fig. 5(a), the cost of DTORA is visibly lower than the other algorithms under the task generation probability of 0.3, 0.5, 0.7, 0.9 and 1. As shown in Fig. 5(c), at a task generation probability of 0.9, the average execution costs for the DTORA, Ly-DPSO, Ly-GA, and Random algorithms are 21.0 ms, 24.1 ms, 25.3 ms, and 29.4 ms, respectively. The DTORA algorithm achieves average execution cost reductions of 12.6%, 16.9%, and 28.4% compared to the Ly-DPSO, Ly-GA, and Random algorithms. Furthermore, a higher task generation probability ρ leads to an increase in the average execution cost for all algorithms. For example, in Fig. 5(b), when the reliability constraint is fixed at 0.90, the cost of the DTORA algorithm is 4.3 ms, 8.5 ms, 14.2 ms, 20.8 ms, and 25.6 ms as ρ is set to 0.3, 0.5, 0.7, 0.9, and 1, respectively. As the reliability constraints become stricter, the average execution cost increases for all algorithms. For example, in Fig. 5(a)(b)(c), when $\rho = 1$, the average execution costs of the DTORA algorithm are 24.9 ms, 25.6 ms, and 26.0 ms as the reliability constraints are set to 0.85, 0.90, and 0.95, respectively.

Fig. 6 presents the runtime of the DTORA, Ly-DPSO, Ly-GA, and C-Opt algorithms in a time slot. Compared to the C-OPT algorithm, the runtime of DTORA, Ly-DPSO, and Ly-GA algorithms are much lower. As ρ increases, the runtime of all the algorithms increases, and the runtime of the DTORA algorithm becomes closer to that of the Ly-DPSO and Ly-GA algorithms. For example, when $\rho = 0.3$, the DTORA algorithm exhibits a relatively higher overhead compared to the Ly-GA and Ly-DPSO algorithms, with a noticeable performance gap. As ρ increases, this gap decreases. For example, when $\rho = 0.9$, the runtime of DTORA, Ly-DPSO, and Ly-GA are 0.174 ms, 0.170 ms, and 0.165 ms, respectively, and DTORA is only 2.2% and 5.2% slightly higher than that of the Ly-DPSO and Ly-GA algorithms, respectively, and accounts for merely 8.7% of the entire time slot. As ρ increases, the runtime of the C-Opt algorithm becomes increasingly distinct compared to other algorithms. For example, when $\rho = 0.9$, the runtime of C-Opt is approximately 16 times longer than that of the DTORA algorithm, making it unsuitable for dynamic systems. Thus, due to the short runtime of the algorithm, the energy consumption of DTORA is negligible.

Summary of Figs. 4 and 6: although the runtime of DTORA is slightly higher than that of Ly-DPSO and Ly-GA, the cost is obviously lower compared to Ly-DPSO and Ly-GA. In contrast, while the C-Opt algorithm demonstrates the lowest cost, its runtime is unacceptable. However, our DTORA algorithm achieves satisfactory performance in terms of both cost and runtime.

5.3.2. The impact of parameters

Fig. 7 presents the execution cost of the DTORA algorithm for different values of sub-populations $\gamma = \{2, 3, 4, 5, 6, 7, 8\}$. It is evident that $\gamma = 3$ results in the lowest cost, indicating the best performance of DTORA. This is because a larger γ allows sub-populations to evolve in various directions, facilitating the exploration of multiple local optima. However, this comes with a trade-off: increasing γ reduces the number of particles per sub-population. Thus, DTORA obtains a higher cost at $\gamma = 7, 8$.

Fig. 8 illustrates the battery energy of EDs controlled by different perturbation parameters θ, when the number of EDs is 30. The dotted lines illustrate the upper bound of battery energy for EDs with the

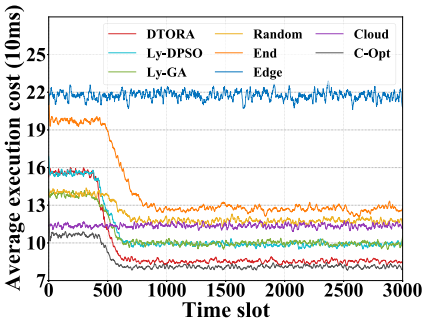


Fig. 10. Average execution costs of different algorithms in large-scale systems.

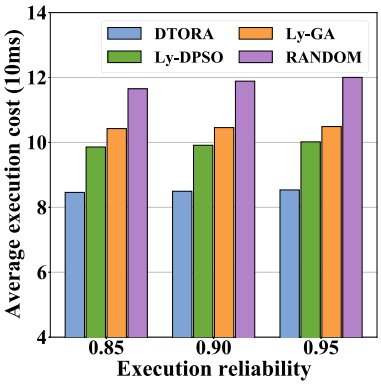


Fig. 11. Average execution costs of different algorithms under varying execution reliability in large-scale systems.

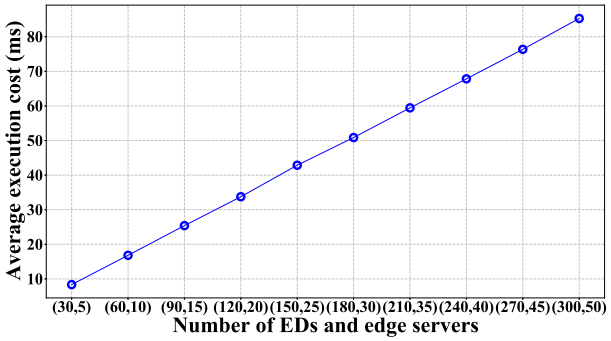


Fig. 12. Average execution costs under varying numbers of EDs and edge servers in large-scale systems.

value of $\theta_n + E_H^{\max}$. During the initial slots, the EDs utilize the harvested energy to charge their batteries, leading to a linear increase in battery energy. As shown in Fig. 8(a), after stabilization, ED D_6 achieves a higher battery energy. This discrepancy arises from the differences in θ_n for each ED D_n, resulting in distinct virtual energy queues and consequently different stable battery energy. Furthermore, after the 1000th slot, all EDs stabilize at their respective upper energy bounds. Battery energy remains constrained within the range $[0, \theta_n + E_H^{\max}]$, confirming the validity of Theorem 3.

Fig. 9 demonstrates the effects of the control parameter V and the lower bound on battery output energy $E^{\min}$ on battery energy and average execution cost. The parameter settings of V and $E^{\min}$ for our experimental evaluation are determined following prior studies [15, 20]. The change of the average execution cost is depicted in Fig. 9(a). As the value of $E^{\min}$ decreases, the average execution cost increases. For $V = 1.0e-4$ and $E^{\min} = 0.01$ mJ, the average execution cost is highest during the first 800 time slots. This is because $E^{\min}$ influences the minimum CPU frequency for local execution, resulting in slower task execution at the initial stage. In later stages, the stabilized average execution cost decreases as V increases or $E^{\min}$ decreases. For instance, when $E^{\min} = 0.02$ mJ, the stabilized cost is highest for $V = 0.5e-4$ and lowest for $V = 1.6e-4$. This is due to the combined effect of V and $E^{\min}$ on the penalty term λ in Eq. (21), encouraging energy-efficient task execution over task dropping.

The change of battery energy is shown in Fig. 9(b). The results indicate that higher V values or lower $E^{\min}$ values lead to higher stabilized battery energy. For example, when $V = 1.0e-4$ and $E^{\min} = 0.01$ mJ, the stabilized battery energy is nearly double that of $V = 1.0e-4$ and $E^{\min} = 0.02$ mJ. Similarly, the battery energy for $V = 1.0e-4$ and $E^{\min} = 0.02$ mJ is approximately twice that of $V = 0.5e-4$ and $E^{\min} = 0.02$ mJ. It is obvious that the stabilized battery energy is directly proportional to V and inversely proportional to $E^{\min}$, as validated by Eq. (21). Therefore, by selecting appropriate combinations of $(V, E^{\min})$, it is possible to adapt the system with various performance requirements.

5.3.3. Large-scale system simulations

To evaluate the feasibility of the algorithm in large-scale systems, we conduct a set of simulation experiments involving 300 EDs and 50 edge servers. As shown in Fig. 10, when $\rho = 0.5$ and the resource configuration is Medium, the stabilization costs for the Edge, End, Random, Cloud, Ly-GA, Ly-DPSO, DTORA, and C-Opt algorithms are 217.4 ms, 127.1 ms, 117.4 ms, 113.7 ms, 99.6 ms, 98.8 ms, and 85.3 ms, respectively. Compared to these algorithms, DTORA reduces the execution cost by 60.8%, 33.9%, 27.4%, 25.0%, 14.4%, and 13.7%, respectively. Fig. 11 shows the performance of these algorithms on the average execution time under varying reliability constraints for task execution. It can be seen that our algorithm consistently achieves the lowest execution cost compared to other algorithms. The experimental results indicate that our DTORA maintains robust performance even in large-scale systems. Fig. 12 presents the results of average execution time under varying numbers of EDs and edge servers, showing that execution time increases linearly with the number of EDs and edge servers. This observation validates the algorithmic complexity and scalability analysis discussed in Section 4.6. In our future real-world experiments, we can implement Jetson Orin Nano boards featuring a 6-core ARM Cortex-A78AE architecture as EH EDs, a Dell P5820 workstation to simulate edge servers, and an NVIDIA RTX 3090 to simulate a cloud server. Each ED can be equipped with a solar panel, an EH module, and a battery. To facilitate energy monitoring in experiments, we can integrate Raspberry Pi 4B units with voltage/power monitors to track harvested energy.

6. Conclusion and future work

In this paper, we have proposed a new task offloading and resource allocation method for EH-EEC computing systems. Our approach has addressed the critical challenges of resource constraints, task execution reliability, and battery energy stability. We have formulated the task offloading problem as a cost optimization problem, considering factors such as ED capacity, task reliability, and energy consumption. By leveraging Lyapunov optimization, we have provided optimal closed-form solutions for computation and transmission power allocation. Additionally, we have designed the MDPSO algorithm to determine the optimal offloading strategy. Extensive experimental results have demonstrated the superiority of our method in terms of reducing delay, enhancing task execution reliability, and maintaining energy stability under dynamic conditions.

In future work, we plan to extend our scheduling framework to accommodate more complex edge-cloud systems, particularly those with temporally correlated energy harvesting patterns. This involves

developing models to capture time-dependent energy arrival and designing robust scheduling policies for such scenarios. Additionally, we will also adapt our solution to specific EH models, such as those related to solar energy.

CRediT authorship contribution statement

Xiaozhu Song: Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Investigation, Formal analysis. **Qianpiao Ma:** Writing – review & editing, Validation, Supervision, Formal analysis, Data curation, Conceptualization. **Gan Zheng:** Writing – review & editing. **Liying Li:** Formal analysis, Methodology, Writing – review & editing. **Peijin Cong:** Writing – review & editing. **Junlong Zhou:** Writing – review & editing, Validation, Supervision, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization.

Declaration of competing interest

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

Appendix A. Proof for Theorem 2

Recall that the proposed DTORA algorithm can achieve the minimization of problem $\mathbb{P}3$ in each slot. We use Π to represent a particular solution. According to Eq. (27), the following inequality holds:

$$\begin{aligned}\Delta_V(t) =& \Delta(t) + V \cdot \mathbb{E}\left[cost^t | \tilde{Q}^t\right]\\ \leq& \Phi + \sum_{n=1}^{N} \mathbb{E}\left[\tilde{Q}_n^t \left(e_n^t - E_n^{t,*}\right) + V \cdot cost_n^{t,*} | \tilde{Q}^t\right]\\ \leq& \Phi + \sum_{n=1}^{N} \mathbb{E}\left[\tilde{Q}_n^t \left(e_n^t - E_n^{t,\Pi}\right) + V \cdot cost_n^{t,\Pi} | \tilde{Q}^t\right]\\ \overset{(\dagger)}{\leq}& \Phi + \upsilon \sum_{n=1}^{N} \tilde{Q}_n^t + V(cost^{\text{opt}} + \upsilon)\\ \overset{(\ddagger)}{\leq}& \Phi + \upsilon \sum_{n=1}^{N} \max\left\{\theta_n, E_H^{\max}\right\} + V(cost^{\text{opt}} + \upsilon).\end{aligned}$$

Inequality (†) holds because our policy Π operates independently of the virtual battery level $\tilde{Q}_n^t$. Furthermore, inequality (‡) is derived by applying Theorem 1 in conjunction with Lemma 2.

Lemma 2. *For $\forall \upsilon > 0$, there exists a static and randomized strategy Π, applicable to $\mathbb{P}3$, that satisfies the following inequality:*

$$\sum_{i=1}^{N} \mathbf{E}\left[cost_n^{t,\Pi}\right] \leq cost^{\text{opt}} + \upsilon,$$

$$\left|\mathbf{E}\left[e_n^t - E_n^{t,\Pi}\right]\right| \leq \upsilon,$$

where $cost^{\text{opt}} = \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \sum_{n=1}^{N} \mathbb{E}\left[cost_n^{t,\text{opt}}\right]$ is the theoretically optimal cost corresponding to the optimal decision of the system to problem $\mathbb{P}2$.

Proof. This proof can be obtained by Theorem 4.5 in [51], which is omitted for brevity. □

When $\upsilon \to 0$, we have

$$\Delta_V(t) \leq \Phi + V cost^{\text{opt}}.$$

Summing over all time slots and taking the average gives

$$\frac{1}{T}\mathbb{E}\left[(\mathscr{L}(t) - \mathscr{L}(t-1)) + \cdots + (\mathscr{L}(1) - \mathscr{L}(0)) | \tilde{Q}^t\right] + \frac{V}{T} \sum_{t=0}^{T-1} \sum_{n=1}^{N} \mathbb{E}\left[cost_n^{t,*}\right] \leq \Phi + V cost^{\text{opt}}.$$

Since $\mathscr{L}(0) = 0$, as $T \to \infty$, we can obtain the upper bound of the system's long-term average execution cost:

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \sum_{n=1}^{N} \mathbb{E}\left[cost_n^{t,*}\right] \leq \frac{\Phi}{V} + cost^{\text{opt}}.$$

Then we will get the upper boundary of long-term net harvesting energy. Assume there exists $\varrho > 0$ and function $\Omega(\varrho)$ that satisfies the policy β, then we have

$$\sum_{i=1}^{N} \mathbf{E}\left[cost_n^{t,\beta}\right] = \Omega(\varrho),$$

$$\mathbf{E}\left[e_n^t - E_n^{t,\Pi}\right] \leq -\varrho.$$

Substituting into Eq. (27), we have

$$\begin{aligned}\Delta_V(t) =& \Delta(t) + V \cdot \sum_{n=1}^{N} \mathbb{E}\left[cost_n^{t,*} | \tilde{Q}^t\right]\\ \leq& \Phi + V\Omega(\varrho) - \varrho \sum_{i=1}^{N} \tilde{Q}^t.\end{aligned}$$

Summing over all time slots and taking the average gives

$$\begin{aligned}&\frac{1}{T} \sum_{t=0}^{T-1} \sum_{n=1}^{N} \mathbb{E}\left[\tilde{Q}^t\right]\\ \leq& \frac{1}{\varrho}\left\{\Phi + V(\Omega(\varrho)) - \frac{1}{T} \sum_{t=0}^{T-1} \sum_{n=1}^{N} \mathbb{E}\left[cost_n^{t,*}\right]\right\}\\ \leq& \frac{1}{\varrho}\left\{\Phi + V(cost^{\max} - cost^{\text{opt}})\right\}.\end{aligned}$$

Since $\sum_{t=0}^{T-1} \sum_{n=1}^{N} \mathbb{E}\left[\tilde{Q}^t\right] \geq \sum_{t=0}^{T-1} \sum_{n=1}^{N} \mathbb{E}\left[e_n^t - E_n^t\right]$, we have the upper boundary of long-term net harvesting energy:

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \sum_{n=1}^{N} \mathbb{E}\left[e_n^t - E_n^{t,*}\right] \leq \Phi + V(cost^{\max} - cost^{\text{opt}}).$$

Appendix B. Proof for Lemma 1

First prove that $F(p_n^t)$ is monotonically increasing about p_n^t. Letting $\chi = h_{mn}^t p_n^t \sigma^{-2}$, the function can be deformed as

$$F_1(\chi) = \frac{\sigma^2 \xi \chi}{A_n^t \log_2(1+\chi)} = \frac{\sigma^2 \xi \ln 2}{A_n^t} \cdot \frac{\chi}{\ln(1+\chi)}.$$

Taking the first order derivative of $F_1(\chi)$, we get

$$F_1'(\chi) = \frac{\sigma^2 \xi \ln 2}{A_n^t} \cdot \frac{(1+\chi)\ln(1+\chi) - \chi}{(1+\chi)\left[\ln(1+\chi)\right]^2}.$$

When $\chi > 0$, it is easy to see that the denominator of $F_1'(\chi)$ is greater than zero. Let $F_2(\chi) = (1+\chi)\ln(1+\chi) - \chi$. Then for the derivative of $F_2(\chi)$, we have

$$F_2'(\chi) = \ln(1+\chi).$$

Since $F_2(0) = 0$, for $\chi > 0$, the numerator of $F_1'(\chi)$ is positive. This implies that $F_1(\chi)$ is monotonically increasing when $\chi > 0$. Given that $p_n^t > 0$, it follows that $F(p_n^t)$ is monotonically increasing with respect to p_n^t. Therefore, the minimum value of $F(p_n^t)$ occurs at $p_n^t = 0$. Now, we focus on solving the limit

$$\lim_{p_n^t \to 0} \frac{\xi h_{mn}^t p_n^t}{A_n^t \log_2 \varepsilon},$$

where $\varepsilon = 1 + h_{mn}^t p_n^t \sigma^{-2}$. Using L'Hospital's rule [52], the numerator and denominator are simultaneously derived for p_n^t to give

$$\lim_{p_n^t \to 0} \frac{\xi h_{mn}^t}{\frac{A_n^t h_{mn}^t}{\sigma^2 \xi \ln 2}} = \sigma^2 \xi {A_n^t}^{-1} \ln 2.$$

In the end, the range of values of function $F(p_n^t)$ is $\left(\sigma^2 \xi {A_n^t}^{-1} \ln 2, +\infty\right)$.

Data availability

Data will be made available on request.

References

[1] A. Wright, Worldwide IDC Global DataSphere Forecast, 2024–2028: AI Everywhere, But Upsurge in Data Will Take Time, Tech. Rep., Global DataSphere, 2024, [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=US52076424.

[2] J. Mei, L. Dai, Z. Tong, X. Deng, K. Li, Throughput-aware dynamic task offloading under resource constant for MEC with energy harvesting devices, IEEE Trans. Netw. Serv. Manag. 20 (3) (2023) 3460–3473.

[3] H. Jiang, X. Dai, Z. Xiao, A. Iyengar, Joint task offloading and resource allocation for energy-constrained mobile edge computing, IEEE Trans. Mob. Comput. 22 (7) (2023) 4000–4015.

[4] L. Wu, P. Sun, Z. Wang, Y. Li, Y. Yang, Computation offloading in multi-cell networks with collaborative edge-cloud computing: A game theoretic approach, IEEE Trans. Mob. Comput. 23 (3) (2024) 2093–2106.

[5] J. Zhang, Z. Ning, R.H. Ali, M. Waqas, S. Tu, I. Ahmad, A many-objective ensemble optimization algorithm for the edge cloud resource scheduling problem, IEEE Trans. Mob. Comput. 23 (2) (2024) 1330–1346.

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

[7] Q. Ma, H. Xu, H. Wang, Y. Xu, Q. Jia, C. Qiao, Fully distributed task offloading in vehicular edge computing, IEEE Trans. Veh. Technol. 73 (4) (2024) 5630–5646.

[8] B. Kar, W. Yahya, Y.-D. Lin, A. Ali, Offloading using traditional optimization and machine learning in federated cloud–edge–fog systems: A survey, IEEE Commun. Surv. & Tutorials 25 (2) (2023) 1199–1226.

[9] U. Saleem, Y. Liu, S. Jangsher, X. Tao, Y. Li, Latency minimization for D2D-enabled partial computation offloading in mobile edge computing, IEEE Trans. Veh. Technol. 69 (4) (2020) 4472–4486.

[10] J. Yang, Z. Guo, J. Luo, Y. Shen, K. Yu, Cloud-edge-end collaborative caching based on graph learning for cyber-physical virtual reality, IEEE Syst. J. 17 (4) (2023) 5097–5108.

[11] J. Tang, T. Qin, Y. Xiang, Z. Zhou, J. Gu, Optimization search strategy for task offloading from collaborative edge computing, IEEE Trans. Serv. Comput. 16 (3) (2023) 2044–2058.

[12] J. Zhou, K. Cao, X. Zhou, M. Chen, T. Wei, S. Hu, Throughput-conscious energy allocation and reliability-aware task assignment for renewable powered in-situ server systems, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 41 (3) (2021) 516–529.

[13] J. Zhou, X.S. Hu, Y. Ma, J. Sun, T. Wei, S. Hu, Improving availability of multicore real-time systems suffering both permanent and transient faults, IEEE Trans. Comput. 68 (12) (2019) 1785–1801.

[14] J. Zhou, T. Wei, M. Chen, X.S. Hu, Y. Ma, G. Zhang, J. Yan, Variation-aware task allocation and scheduling for improving reliability of real-time mpsocs, in: Design, Automation & Test in Europe Conference & Exhibition, DATE, 2018, pp. 171–176.

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

[16] Z. Chang, L. Liu, X. Guo, Q. Sheng, Dynamic resource allocation and computation offloading for IoT fog computing system, IEEE Trans. Ind. Informatics 17 (5) (2021) 3348–3357.

[17] F. Zhao, Y. Chen, Y. Zhang, Z. Liu, X. Chen, Dynamic offloading and resource scheduling for mobile-edge computing with energy harvesting devices, IEEE Trans. Netw. Serv. Manag. 18 (2) (2021) 2154–2165.

[18] G. Zhang, W. Zhang, Y. Cao, D. Li, L. Wang, Energy-delay tradeoff for dynamic offloading in mobile-edge computing system with energy harvesting devices, IEEE Trans. Ind. Informatics 14 (10) (2018) 4642–4655.

[19] Y. Mao, J. Zhang, K.B. Letaief, A Lyapunov optimization approach for green cellular networks with hybrid energy supplies, IEEE J. Sel. Areas Commun. 33 (12) (2015) 2463–2477.

[20] K. Zeng, X. Li, T. Shen, Energy-stabilized computing offloading algorithm for uavs with energy harvesting, IEEE Internet Things J. 11 (4) (2024) 6020–6031.

[21] X. Hou, J. Zhou, L. Li, M. Zhao, P. Cong, Z. Wu, S. Hu, ILRM: Imitation learning based resource management for integrated CPU-GPU edge systems with renewable energy sources, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 44 (6) (2024) 2392–2397.

[22] G. Gao, M. Xiao, J. Wu, H. Huang, S. Wang, G. Chen, Auction-based VM allocation for deadline-sensitive tasks in distributed edge cloud, IEEE Trans. Serv. Comput. 14 (6) (2021) 1702–1716.

[23] Z. Hu, C. Fang, Z. Wang, S.-M. Tseng, M. Dong, Many-objective optimization based-content popularity prediction for cache-assisted cloud-edge-end collaborative IoT networks, IEEE Internet Things J. 11 (1) (2024) 1190–1200.

[24] H. Zhou, Z. Zhang, D. Li, Z. Su, Joint optimization of computing offloading and service caching in edge computing-based smart grid, IEEE Trans. Cloud Comput. 11 (2) (2023) 1122–1132.

[25] H. Xiao, J. Huang, Z. Hu, M. Zheng, K. Li, Collaborative cloud-edge-end task offloading in MEC-based small cell networks with distributed wireless backhaul, IEEE Trans. Netw. Serv. Manag. 20 (4) (2023) 4542–4557.

[26] Y. Chen, J. Zhao, Y. Wu, J. Huang, X. Shen, Qoe-aware decentralized task offloading and resource allocation for end-edge-cloud systems: A game-theoretical approach, IEEE Trans. Mob. Comput. 23 (1) (2024) 769–784.

[27] N. Sharma, A. Ghosh, R. Misra, S.K. Das, Deep meta q-learning based multi-task offloading in edge-cloud systems, IEEE Trans. Mob. Comput. 23 (4) (2024) 2583–2598.

[28] H. Yuan, M. Zhou, Profit-maximized collaborative computation offloading and resource allocation in distributed cloud and edge computing systems, IEEE Trans. Autom. Sci. Eng. 18 (3) (2021) 1277–1287.

[29] T. Tang, C. Li, F. Liu, Collaborative cloud-edge-end task offloading with task dependency based on deep reinforcement learning, Comput. Commun. 209 (2023) 78–90.

[30] W. Fan, X. Liu, H. Yuan, N. Li, Y. Liu, Time-slotted task offloading and resource allocation for cloud-edge-end cooperative computing networks, IEEE Trans. Mob. Comput. 23 (8) (2024) 8225–8241.

[31] F. Chen, J. Zhou, X. Xia, Y. Xiang, X. Tao, Q. He, Joint optimization of coverage and reliability for application placement in mobile edge computing, IEEE Trans. Serv. Comput. 16 (6) (2023) 3946–3957.

[32] A. Al-Dulaimy, C. Sicari, A.V. Papadopoulos, A. Galletta, M. Villari, M. Ashjaei, Tolerancer: A fault tolerance approach for cloud manufacturing environments, in: 2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation, ETFA, IEEE, 2022, pp. 1–8.

[33] Y. Qiu, J. Liang, V.C. Leung, X. Wu, X. Deng, Online reliability-enhanced virtual network services provisioning in fault-prone mobile edge cloud, IEEE Trans. Wirel. Commun. 21 (9) (2022) 7299–7313.

[34] J. Li, W. Liang, M. Huang, X. Jia, Reliability-aware network service provisioning in mobile edge-cloud networks, IEEE Trans. Parallel Distrib. Syst. 31 (7) (2020) 1545–1558.

[35] R. Lin, Z. Zhou, S. Luo, Y. Xiao, X. Wang, S. Wang, M. Zukerman, Distributed optimization for computation offloading in edge computing, IEEE Trans. Wirel. Commun. 19 (12) (2020) 8179–8194.

[36] X. Chen, Decentralized computation offloading game for mobile cloud computing, IEEE Trans. Parallel Distrib. Syst. 26 (4) (2015) 974–983.

[37] J. Zhou, X. Hou, Y. Zeng, P. Cong, W. Jiang, S. Guo, Quality of experience and reliability-aware task offloading and scheduling for multi-user mobile-edge computing systems, IEEE Trans. Serv. Comput. (2025) 1–14.

[38] T.X. Tran, D. Pompili, Joint task offloading and resource allocation for multi-server mobile-edge computing networks, IEEE Trans. Veh. Technol. 68 (1) (2019) 856–868.

[39] M.R. Jan, C. Anantha, N. Borivoje, et al., Digital integrated circuits: a design perspective, Pearson (2003).

[40] C. Kai, H. Zhou, Y. Yi, W. Huang, Collaborative cloud-edge-end task offloading in mobile-edge computing networks with limited communication capability, IEEE Trans. Cogn. Commun. Netw. 7 (2) (2020) 624–634.

[41] X. Tian, H. Meng, Y. Li, P. Miao, P. Xu, Dynamic computation offloading for green things-edge-cloud computing with local caching, in: IEEE International Parallel and Distributed Processing Symposium, IPDPS, 2022, pp. 1018–1028.

[42] D. Zhu, R. Melhem, D. Mossé, The effects of energy management on reliability in real-time embedded systems, in: IEEE/ACM International Conference on Computer Aided Design, 2004, pp. 35–40.

[43] L. Huang, M.J. Neely, Utility optimal scheduling in energy harvesting networks, in: Proceedings of the Twelfth ACM International Symposium on Mobile Ad Hoc Networking and Computing, 2011, pp. 1–11.

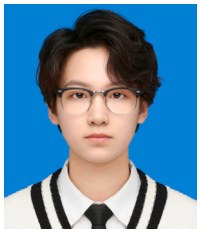
[44] Y. Mao, J. Zhang, K.B. Letaief, ARQ with adaptive feedback for energy harvesting receivers, in: 2016 IEEE Wireless Communications and Networking Conference, IEEE, 2016, pp. 1–6.

[45] M.J. Neely, L. Huang, Dynamic product assembly and inventory control for maximum profit, in: 49th IEEE Conference on Decision and Control, CDC, 2010, pp. 2805–2812.

[46] I.B. Vapnyarskii, Lagrange multipliers, 2001, [1994], Encyclopedia of Mathematics, EMS Press. [Online]. Available: https://encyclopediaofmath.org/wiki/Lagrangian.

[47] S. Strasser, R. Goodman, J. Sheppard, S. Butcher, A new discrete particle swarm optimization algorithm, in: Proceedings of the Genetic and Evolutionary Computation Conference 2016, 2016, pp. 53–60.

[48] B. Liang, Y. Zhao, Y. Li, A hybrid particle swarm optimization with crisscross learning strategy, Eng. Appl. Artif. Intell. 105 (2021) 104418.
[49] Z. Tong, J. Cai, J. Mei, K. Li, K. Li, Dynamic energy-saving offloading strategy guided by Lyapunov optimization for IoT devices, IEEE Internet Things J. 9 (20) (2022) 19903–19915.
[50] H. Zhao, W. Du, W. Liu, T. Lei, Q. Lei, QoE aware and cell capacity enhanced computation offloading for multi-server mobile edge computing systems with energy harvesting devices, in: 2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), IEEE, 2018, pp. 671–678.
[51] M.J. Neely, Stochastic network optimization with application to communication and queueing systems, Synth. Lect. Commun. Networks (2010) 1–211, [Online]. Available: http://dx.doi.org/10.2200/s00271ed1v01y201006cnt007.
[52] E.W. Weisstein, L'hospital's rule, 2003, [Online]. Available: https://mathworld.wolfram.com/.

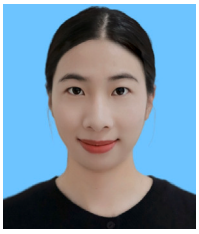


Xiaozhu Song received the B.S. degree in Computer Science and Technology from the Nanjing University of Science and Technology, Nanjing, China, in 2024. She is currently pursuing her M.S. degree with Nanjing University of Science and Technology, Nanjing, China. Her research interests include edge computing, and federated learning.

Qianpiao Ma received the B.S. degree in computer science and the Ph.D. degree in computer software and theory from the University of Science and Technology of China, Hefei, China, in 2014 and 2022, respectively. He is currently an Associate Professor at the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China. His primary research interests include federated learning, mobile-edge computing, and distributed machine learning.

Gan Zheng is a senior engineer with the Big Data Technology Development Department of the Guangxi Zhuang Autonomous Region Information Center. His research interests are in the area of big data, edge computing and IoT, where he has published a dozen of refereed papers.

Liying Li received her 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 Nanjing University of Science and Technology, Nanjing, China. Her current research interests are in the areas of cyber–physical systems, IoT resource management, 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 at the School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China. Her research interests are in the areas of cloud computing, service computing, and IoT, where she has published 30 refereed papers, more than half of which appeared in IEEE/ACM Transaction papers.

Junlong Zhou received a 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, during 2014–2015. He is currently an Associate Professor at the Nanjing University of Science and Technology, Nanjing, China. His research interests include edge computing, cloud computing, and embedded systems, where he has published 120 refereed papers, including over 40 in premier IEEE/ACM Transactions. He has been an Associate Editor for the Journal of Sustainable Computing: Informatics and Systems, the Journal of Circuits, Systems, and Computers, and the IET Cyber–Physical Systems: Theory & Applications, a Subject Area Editor for the Journal of Systems Architecture: Embedded Software Design, and a Guest Editor for 8 Journals such as ACM Transactions on Cyber–Physical Systems. He received the Best Paper Awards from IEEE iThings 2020, IEEE CPSCom 2022, and IEEE ICITES 2024.