Spatiotemporal Non-Uniformity-Aware Online Task Scheduling in Collaborative Edge Computing for Industrial Internet of Things

Yang Li[ID], Xing Zhang[ID], *Senior Member, IEEE,* Yukun Sun, Wenbo Wang[ID], *Senior Member, IEEE,* and Bo Lei[ID]

Abstract— Mobile edge computing mitigates the shortcomings of cloud computing caused by unpredictable wide-area network latency and serves as a critical enabling technology for the Industrial Internet of Things (IIoT). Unlike cloud computing, mobile edge networks offer limited and distributed computing resources. As a result, collaborative edge computing emerges as a promising technology that enhances edge networks' service capabilities by integrating computational resources across edge nodes. This paper investigates the task scheduling problem in collaborative edge computing for IIoT, aiming to optimize task processing performance under long-term cost constraints. We propose an online task scheduling algorithm to cope with the spatiotemporal non-uniformity of user request distribution in distributed edge networks. For the spatial non-uniformity of user requests across different factories, we introduce a graph model to guide optimal task scheduling decisions. For the time-varying nature of user request distribution and long-term cost constraints, we apply Lyapunov optimization to decompose the long-term optimization problem into a series of real-time subproblems that do not require prior knowledge of future system states. Given the NP-hard nature of the subproblems, we design a heuristic-based hierarchical optimization approach incorporating enhanced discrete particle swarm and harmonic search algorithms. Finally, an imitation learning-based approach is devised to further accelerate the algorithm's operation, building upon the initial two algorithms. Comprehensive theoretical analysis and experimental evaluation demonstrate the effectiveness of the proposed schemes.

Index Terms—Collaborative edge computing, spatiotemporal non-uniformity, graph model, Lyapunov optimization, heuristic-based hierarchical optimization approach, imitation learning.

◆

1 INTRODUCTION

1.1 Research Background and Motivation

MOBILE edge computing (MEC) has emerged as a promising architecture for delivering real-time or low-latency services to users in close proximity. This aligns with the Internet of Things (IoT) trend, where emerging applications often require complex computations and low-latency performance, while IoT devices have limited computational capabilities. Moreover, offloading computational tasks to remote cloud processing introduces significant latency due to long backhaul links. Consequently, edge computing has been proposed as a crucial component of IoT, particularly in Industrial Internet of Things (IIoT) applications. For example, it can be used to verify whether workers are wearing protective gear correctly and to analyze equipment status monitoring [1]. Additionally, edge computing

- *Yang Li, Xing Zhang, Yukun Sun and Wenbo Wang are with the School of Information and Communications Engineering, Beijing University of Posts and Telecommunications, Beijing 100876, China. E-mail: ly209991@bupt.edu.cn, zhangx@ieee.org, sunyukun@bupt.edu.cn, wbwang@bupt.edu.cn.*
- *Bo Lei is with Beijing Branch of China Telecom Co., Ltd., Beijing 100032, China. E-mail: leibo@chinatelecom.cn.*

addresses privacy concerns in IIoT, such as managing confidential product data that may contain sensitive information.

A typical form of mobile edge computing is to equip network access points with storage and computing capabilities, forming edge nodes. This allows computational tasks to be processed at the network's edge, reducing service response time. Compared to cloud computing, edge nodes possess limited computational resources. Meanwhile, the increasing service demand from end devices may overload MEC installations, while cost constraints restrict the expansion of their computing and storage capabilities. These challenges highlight the need for an end-edge-cloud collaboration framework, where end devices primarily offload computation tasks to the nearby MEC server [2]–[5]. When computational demand exceeds the capacity of the edge node, some tasks can be further offloaded to the cloud. This collaborative architecture facilitates dynamic task allocation and optimizes computing resource utilization, reducing overall system costs while maintaining quality of service (QoS).

While initial research in edge computing has focused on practical topics such as resource allocation, co-scheduling, and latency reduction [6]–[12], certain IIoT applications present unique and fundamentally different challenges. These issues remain largely unexplored but are of significant importance. On one hand, some IIoT applications involve private data that must be processed exclusively by the edge server within the factory [13]. An enterprise

typically operates multiple factories, each of which deploys an edge computing server. In this context, resource allocation and co-scheduling in edge computing are restricted to horizontal collaboration among edge servers, whereas the traditional end-edge-cloud framework emphasizes vertical collaboration. On the other hand, service requests for IIoT devices across different factories vary over time and are spatially non-uniform [14]. Moreover, enterprises typically operate under long-term cost constraints, and task scheduling among edge nodes occurs over the network, incurring additional overhead. Consequently, in IIoT scenarios that involve processing privacy-sensitive data, task scheduling is confined to horizontal collaboration among edge nodes. It must consider both spatial and temporal collaboration to balance task processing efficiency and long-term operational costs.

Nevertheless, satisfying all the aforementioned requirements poses significant challenges for the following reasons.

1) Given the time-varying and stochastic nature of the IIoT system state and service requests, horizontal collaboration strategies must be dynamically adjusted among edge nodes. This necessitates online optimization, where long-term cost budgeting relies on statistical data about future network states and service request distributions, which is difficult, if not impossible, to obtain.

2) The spatial non-uniformity of service requests and resource allocation across different factories necessitates the careful design of horizontal collaboration among edge nodes to enhance task processing efficiency. However, as the number of factories increases, collaborative decisions must account for the resulting computational complexity, since scalability and real-time feasibility are crucial in IIoT scenarios.

3) Spatial and temporal collaboration among edge nodes are interdependent, increasing the complexity of the optimal decision problem and rendering traditional independent optimization methods ineffective.

1.2 Related Work

In edge computing networks, edge nodes offer localized, distributed, and limited computational resources. Due to the resource limitations of individual edge nodes and the dynamic, non-uniform distribution of task processing requests, collaborative edge computing has garnered increasing attention from researchers. In this section, we review publications addressing the task scheduling problem in collaborative edge computing networks.

Several studies have investigated task allocation between multiple users and multiple edge nodes [15]–[17]. In our previous work [15], we proposed a task offloading framework for multi-user, multi-edge servers aimed at minimizing the average task processing latency by jointly optimizing task scheduling, bandwidth, and computational resource allocation at the edge nodes. Tran *et al.* [16] explored the computational offloading problem involving multiple users and edge nodes. They optimized the weighted sum of task completion time and energy consumption, constrained by the condition that each task is processed on only one edge

node. Ma *et al.* [17] introduced a truthful combinatorial double auction mechanism for MEC in IIoT. The mechanism was designed to ensure truthfulness and budget balance under locality constraints by optimizing resource allocation and pricing. However, these studies assume that end devices can directly access all edge nodes, making them unsuitable for IIoT scenarios with restricted access ranges. In practice, IIoT devices in a factory can only transmit data to the edge node within the same factory, not to edge nodes in other locations. Consequently, tasks must be redistributed among collaborating edge nodes via the network.

Several studies have explored how edge nodes can further offload tasks to collaborative nodes, ensuring acceptable latency and cost [10], [18], [19]. Lee *et al.* [10] introduced a task scheduling algorithm that prioritizes tasks based on their deadlines and optimizes network flow allocation to minimize the deadline miss ratio of applications. Ebrahimzadeh *et al.* [18] examined collaborative edge computation in FiWi-enhanced HetNets, where Optical Network Units (ONU) are equipped with edge computing servers. The central Optical Line Terminal (OLT), located at a distance from the ONU in the fiber backhaul, is also equipped with computation servers that assist the ONU in further offloading tasks. Hou *et al.* [19] studied task scheduling in collaborative edge computing, using incentives to encourage the allocation of tasks to the optimal computing node for processing. However, these studies focus solely on short-term performance gains, neglecting the effects of fluctuations caused by the time-varying distribution of user requests on long-term utility.

Online optimization is a powerful technique for managing dynamic MEC systems with inherent uncertainties while ensuring long-term performance. Several studies have applied deep reinforcement learning (DRL) to MEC resource management and task scheduling optimization [20]–[23]. However, the DRL training process may encounter policy instability, over-exploration, or oscillations, leading to unstable system performance or even failure to meet QoS requirements [24]. Additionally, as a black-box model, DRL lacks strict performance guarantees. To address these challenges, Lyapunov-based online optimization offers a viable alternative. The Lyapunov-based online optimization approach offers an efficient, stable, and interpretable framework for MEC resource management, optimizing performance while ensuring long-term system stability [25]. Existing research primarily focuses on single-timescale online decision-making based on Lyapunov optimization [25]–[27] or extends to two-timescale joint online decision-making [28]–[30]. However, these studies focus solely on optimization along the time dimension and cannot be directly applied to the problem addressed in this paper. Due to the spatiotemporal non-uniformity of service requests from IIoT devices in different factories, the optimization in the time and spatial dimensions are inherently coupled.

In recent years, the use of graph models in network optimization has garnered significant attention, particularly in areas such as routing optimization [31], [32] and resource allocation [22], [33], [34]. Studies in [31] and [32] represented the network as a graph and employed graph theory algorithms to identify the shortest paths. In [22], services were modeled as directed acyclic graphs (DAGs) to optimize task

TABLE 1
A Table Comparing Our Work With The Existing Studies

Reference	Method	Task Scheduling Model	Consideration of Time-Varying Nature	Consideration of Spatial Non-Uniformity	Prototype Implementation and Validation
[15], [16]	Low-complexity heuristic approach	Users directly offload tasks to the corresponding MEC servers.	No	No	No
[17]	Combinatorial double auction-based algorithm	Users directly offload tasks to the corresponding MEC servers.	No	No	No
[18]	Self-organization based mechanism	Tasks are redistributed among edge nodes via the network.	No	Yes	No
[19]	Incentive-based approach	Tasks are redistributed among edge nodes and auxiliary devices.	No	Yes	No
[10]	Low-complexity heuristic approach	Tasks are redistributed among edge nodes via the network.	No	Yes	No
[20]–[23]	DRL-based algorithm	Tasks are divided into multiple subtasks and scheduled in parallel to multiple edge servers for processing.	Yes	No	No
[25]	Lyapunov-based single-timescale online optimization algorithm	Tasks are redistributed among edge nodes via the network.	Yes	No	No
[26], [27]	Lyapunov-based single-timescale online optimization algorithm	Tasks are redistributed between edge nodes and the cloud.	Yes	No	No
[28], [29]	Lyapunov-based two-timescale online optimization algorithm	Users directly offload tasks to the corresponding MEC servers.	Yes	No	No
[30]	Lyapunov-based two-timescale online optimization algorithm	Tasks are redistributed among edge nodes via the network.	Yes	No	No
[31], [32]	Shortest path algorithm	Tasks are transmitted according to the specified path.	No	No	No
[33], [34]	Bipartite graph matching algorithm	Users directly transmit tasks to the computing nodes.	No	No	No
Proposed work	A approach integrates Lyapunov optimization with the graph-based model	Tasks are redistributed among edge nodes via the network.	Yes	Yes	Yes

partitioning. Studies in [33] and [34] represented the relationships between tasks and computing nodes as bipartite graphs and applied graph theory algorithms to solve the optimal matching problem. However, these studies also do not consider the spatiotemporal non-uniformity of service request distribution and suffer from high computational costs in large-scale network environments. In contrast to the above research, we leverage graph models to guide task scheduling, narrow the search space for decision-making, and employ a graph neural network for imitation learning to further reduce algorithm runtime.

In summary, different from all existing work (as demonstrated in Table 1), this paper proposes a novel spatiotemporal non-uniformity-aware online task scheduling scheme for collaborative edge computing in IIoT. The proposed scheme integrates Lyapunov optimization with a graph-based model to effectively handle the time-varying and spatially non-uniform distribution of service requests and resources. Furthermore, none of the studies discussed in this subsection have demonstrated their superior performance

in real engineering applications, as they are challenging to implement in practice. In this paper, we propose a generic edge node collaboration scheme and develop a prototype testbed using Kubernetes, providing a real-world application case that verifies the superior performance of the proposed scheme.

1.3 Contribution and Organization

The main contributions of this paper are summarized as follows:

- Dynamic task scheduling in the collaborative edge computing network is formulated as a stochastic optimization problem, aiming to optimize long-term average task processing delay under the constraint of long-term operational cost. This problem is NP-hard.
- The Lyapunov optimization is employed to incorporate the long-term task scheduling cost constraint into real-time optimization, allowing task scheduling decisions to be made online without requiring any
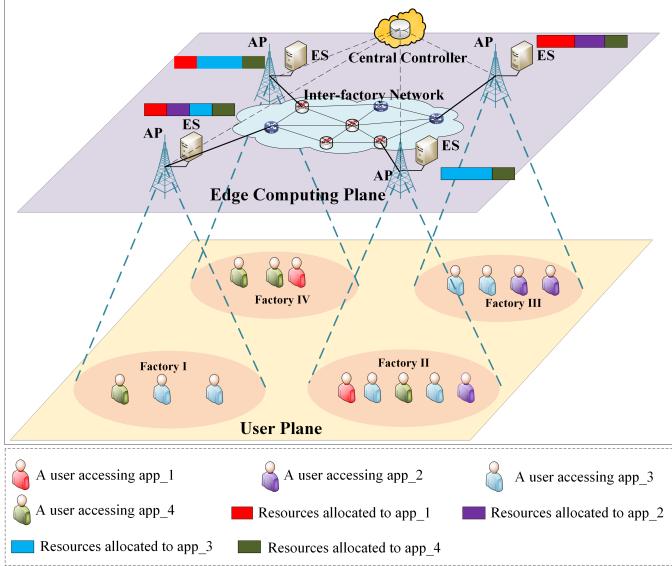
Fig. 1. Illustration of the network architecture for collaborative edge computing in the IIoT

prior knowledge of future information (e.g., changes in the distribution of user requests).

- In terms of the problem derived from the Lyapunov optimization, a graph model is introduced to guide optimal task scheduling decisions, and a two-stage heuristic algorithm is developed.
- To further reduce algorithm execution time, an imitation learning-based scheme is developed based on the previous algorithms. Finally, the performance of the proposed algorithm is demonstrated through theoretical analysis, numerical simulations, and engineering applications.

The remainder of the paper is structured as follows. Section 2 introduces the system model. Section 3 formulates and analyzes the task scheduling problem in collaborative edge computing networks, with the goal of minimizing the long-term average task processing delay, subject to the constraint of long-term operational cost. Section 4 proposes an online task-scheduling framework along with the algorithms implemented within it. Section 5 analyzes the complexity and convergence of the proposed algorithms. Section 6 validates the feasibility and superiority of the proposed algorithms through numerical simulations and practical engineering applications[1]. Section 7 concludes the paper.

2 SYSTEM MODEL

In this section, we first present a comprehensive system description. Subsequently, we detail the considered models, including service request scheduling, response latency, and task processing overhead.

2.1 System Description

As illustrated in Fig. 1, we consider an enterprise with multiple factories located in different geographical regions. For

1. The code is available at https://github.com/CPNGroup/Spatial-Temporal-Non-Uniformity-Aware-Online-Task-Scheduling.

privacy and latency requirements, each factory is equipped with an edge server (ES) at its access point (AP). The APs in all factories are interconnected through the inter-factory network to enable mutual access and task scheduling. Due to the presence of multiple tasks in the factories, services need to be deployed on each ES (as represented by app_1, 2, 3, 4 in Fig. 1), and corresponding resources must be allocated. Resource allocation decisions can be made offline, based on the expected task processing requirements of each factory [35], or online, dynamically adapting to the request distribution of each ES [36]–[38]. Notably, due to the differing time sensitivities of task scheduling and resource allocation, executing these decisions simultaneously would delay task scheduling, violating real-time task requirements. Moreover, this approach would cause the system to orchestrate resources too frequently, increasing operational costs and system instability [23]. Consequently, resource allocation typically operates on a larger timescale than task scheduling. Specifically, resource allocation for each service is determined over a long timescale, and task scheduling occurs over a short timescale [39]. This paper focuses on dynamic task scheduling strategies, assuming that service deployment and resource allocation are completed before task scheduling decisions, aligning with real-world engineering practices.

The IIoT system, as depicted in Fig. 1, can be divided into three parts: the user plane, the edge computing plane, and the central controller. The user plane consists of IIoT devices distributed across different factories, which can initiate requests for different services to the APs of their respective factories. The edge computing plane includes APs, ESs, and the inter-factory network. APs execute task scheduling strategies for different services based on control signals from the central controller. The inter-factory network enables task transmission among factories. ESs hand over received tasks to the services deployed on them for processing. The central controller senses the request distribution and system status of each factory, dynamically adjusts task scheduling strategies for different services, and sends them to the APs for execution.

For simplicity in the following discussion, we unify the notation for ESs, APs, and factories, collectively denoting them as $\mathcal{M} = \{1, 2, \ldots, M\}$. All deployed services are represented as $\mathcal{K} = \{1, 2, \ldots, K\}$. Given the dynamic nature of service request distribution, the system operates in time slots, with time discretized into frames $\mathcal{T} = \{1, 2, \ldots T\}$. Notably, the division of time frames may vary across services, depending on their maximum tolerable delay. To describe the spatiotemporal characteristics of request distribution in the user plane and resource allocation in the edge computing plane, denote $N_m^k(t)$ as the number of kth service requests arriving at the mth AP at the beginning of the tth time slot. Let $F_m^k(t)$ represent the computational resources allocated by ES m to the kth service at the start of the tth time slot. Let $R_{m,m'}^k(t)$ denote the bandwidth allocated for transferring requests of the kth service from AP m to ES m' during the tth time slot, where the first and second subscripts represent the AP and ES indices, respectively. In practice, bandwidth allocation in wired networks based on service demand is common. Since each AP is directly connected to its co-located ES via fiber, we
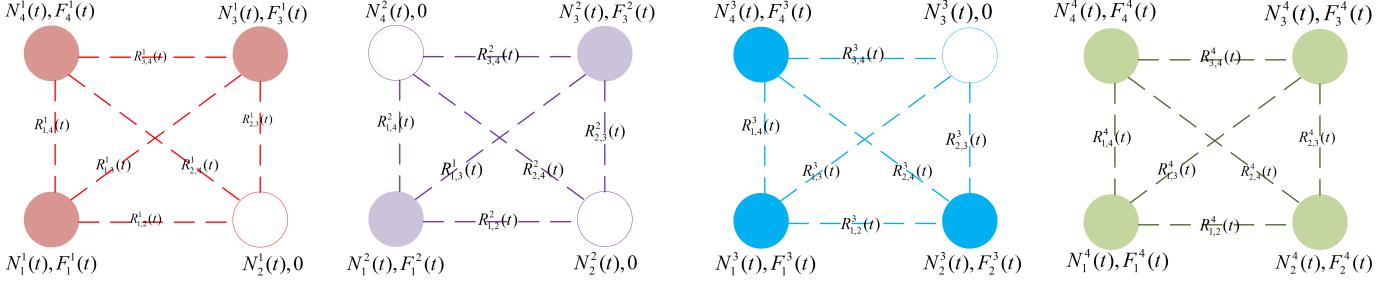
Fig. 2. Computing network graph for each service corresponding to Fig. 1 during the tth time slot. (a) Computing network graph of app_1. (b) Computing network graph of app_2. (c) Computing network graph of app_3. (d) Computing network graph of app_4. Solid and hollow circles represent cases where $F_m^k(t) > 0$ and $F_m^k(t) = 0$, respectively.

assume $R_{m,m}^k(t) = \infty$, i.e., the transmission delay between AP m and ES m is negligible. Using the request distribution from the user side and the resource allocation from the provisioning side, we can model the state of each service at the tth time slot as a computing network graph, as shown in Fig. 2.

Fig. 2 shows that each service's computational network graph is independent, allowing task scheduling decisions to be made separately for each service [40]. To clarify, we will analyze the kth service as an example in the following discussion. The important notations used in the rest of this paper are summarized in Table 2.

TABLE 2
Major Notions

	Symbols	Definition
System Model	$\mathcal{M}$	Set of ESs/APs/factories
	$\mathcal{K}$	Set of services
	$\mathcal{T}$	Set of time slots
	$N_m^k(t)$	Number of requests for app_k arriving at the mth AP at the start of the tth time slot
	$N_{m,m'}^k(t)$	Number of requests for app_k arriving at the mth AP at the start of the tth time slot scheduled for processing on ES m'.
	$F_m^k(t)$	Computing resources allocated by the mth ES to app_k during the tth time slot.
	$R_{m,m'}^k(t)$	Bandwidth allocated for scheduling requests of app_k from AP m to ES m' during the tth time slot.
	$T_k(t)$	Average response delay of all requests for app_k in the tth time slot.
	$E_k(t)$	Task processing cost of all requests for app_k in the tth time slot
	E_k^{avg}	Long-term average task processing cost budget for app_k
	$\mathcal{N}_k$	Online task scheduling decisions for app_k
	$\mathcal{N}_k(t)$	Task scheduling decision for app_k in the tth time slot.
Algorithm	$Q_k(t)$	Virtual control queue length for app_k at the tth time slot.
	V	Weight parameter for balancing the trade-off between the task response latency performance and the task processing cost budget violations
	$\mathcal{O}$	The set of factory/region index for isolated nodes
	$\mathcal{P}$	The set of factory/region index for source nodes
	$\mathcal{Q}$	The set of factory/region index for sink nodes
	NI	Maximum number of iterations for Algorithm 2
	I_{max}	Maximum number of iterations for Algorithm 3

2.2 Service Request Scheduling Model

Fig. 3 shows the processing flow of requests for the kth service initiated by IIoT devices in Factory 1 during the tth time slot. These requests first reach the AP in Factory 1,
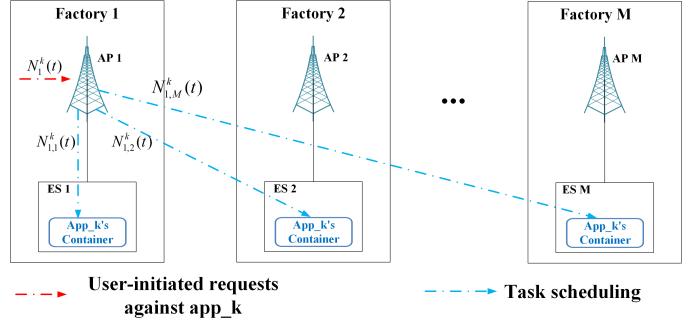

Fig. 3. Illustration of the processing flow for requests against the kth service initiated by IIoT devices in Factory 1

where the task scheduler for the kth service allocates them to ESs hosting the kth service, following the central controller's dispatch policy. Let $N_m^k(t)$ denote the number of requests for the kth service initiated by the IIoT devices in Factory m during the tth time slot. The number of these requests scheduled for processing on ES $m' \in \mathcal{M}$ is represented by $N_{m,m'}^k(t)$, we have:

$$\sum_{m' \in \mathcal{M}} N_{m,m'}^k(t) = N_m^k(t), \tag{1}$$

$$\begin{cases} N_{m,m'}^k(t) = 0 & if \quad F_{m'}^k(t) = 0 \\ N_{m,m'}^k(t) \geq 0 & if \quad F_{m'}^k(t) > 0 \end{cases}. \tag{2}$$

Equation (2) indicates that if ES m' does not allocate resources for the kth service, it will not process any requests for that service. Building on the previously defined task scheduling decisions, the response delay and processing overhead resulting from these decisions can be quantified.

2.3 Response Latency Model

In the considered scenarios, a task's response latency is affected by both scheduling and computation latencies. We exclude task offloading latency and result return latency from consideration, as task scheduling decisions do not impact offloading latency, and the result size is sufficiently small for its return delay to be negligible [22]. Since our focus is on the average response latency of all requests for the kth service during each time slot, we begin by analyzing the total latency of all requests for the kth service in the tth time slot.

- *Scheduling Latency*: Let L_k represent the size of a single request for the kth service. Based on the task scheduling decision, we can calculate the total latency of all requests for the kth service transmitted from AP m to ES m' in the tth time slot as

$$T_{m,m'}^{k,tran}(t) = \frac{N_{m,m'}^k(t)L_k}{R_{m,m'}^k(t)}, \quad (3)$$

where the numerator represents the total data size of all requests for the kth service scheduled from AP m to ES m' in the tth time slot. Thus, the total scheduling delay of all requests for the kth service in the tth time slot can be expressed as

$$T_k^{tran}(t) = \sum_{m\in\mathcal{M}} \sum_{m'\in\mathcal{M}\setminus\{m\}} T_{m,m'}^{k,tran}(t), \quad (4)$$

where the outer summation runs over all APs, and the inner summation is performed over all ESs communicating with the current AP through the inter-factory network.

- *Computation Latency*: Let C_k represent the number of CPU cycles needed to process a single request for the kth service. The total computation time of all tasks for the kth service on ES m in the tth time slot can be calculated as

$$T_m^{k,comp}(t) = \frac{\sum_{m'\in\mathcal{M}} N_{m',m}^k(t)C_k}{F_m^k(t)}, \quad (5)$$

where the numerator represents the total CPU cycles required to process all requests for the kth service that are dispatched from all APs to ES m. Thus, the total computation delay of all tasks for the kth service in the tth time slot can be expressed as

$$T_k^{comp}(t) = \sum_{m\in\mathcal{M}} T_m^{k,comp}(t). \quad (6)$$

Based on the above analyses, the average response delay of all requests for the kth service in the tth time slot can be calculated as

$$T_k(t) = \frac{T_k^{tran}(t) + T_k^{comp}(t)}{\Sigma_{m\in\mathcal{M}} N_m^k(t)}. \quad (7)$$

Specifically, the numerator represents the total response delay of all requests for the kth service in the tth time slot, and the denominator denotes the total number of requests for the kth service initiated by all IIoT devices in the tth time slot.

2.4 Task Processing Cost Model

Optimizing the task scheduling policy among ESs can reduce the average response latency of services. However, task scheduling introduces additional operational costs. For instance, transferring data across regions raises the energy consumption of network devices like routers and switches. Additionally, processing tasks on edge nodes also consume energy. Therefore, consistent with the previous subsection, we categorize task processing overhead into scheduling cost and computation cost.

- *Scheduling Cost*: To model the operational cost of task scheduling, let $E_{m,m'}^k$ represent the cost of scheduling a single request for the kth service from AP m to ES m', where the first and second subscripts represent the AP and ES indices, respectively. Without loss of generality, assume $E_{m,m}^k = 0$, i.e., the transmission cost from AP m to ES m is negligible. Given the task scheduling decision for the kth service in the tth time slot, the task scheduling cost for the kth service in the tth time slot can be calculated as

$$E_k^{tran}(t) = \sum_{m\in\mathcal{M}} \sum_{m'\in\mathcal{M}\setminus\{m\}} N_{m,m'}^k(t)E_{m,m'}^k, \quad (8)$$

where the outer summation runs over all APs, and the inner summation is performed over all ESs communicating with the current AP through the inter-factory network.

- *Computation Cost*: Assuming that all requests for a given service processed on an edge node share the computational resources occupied by that service equally, the computational resources allocated to each request for the kth service on ES m can be expressed as

$$f_m^k(t) = \frac{F_m^k(t)}{\sum_{m'\in\mathcal{M}} N_{m',m}^k(t)}, \quad (9)$$

where the denominator represents the total number of requests for the kth service dispatched from all APs to ES m for processing. Let k_m represent the computational energy efficiency factor of ES m. According to [29] and [41], the total computational energy consumption of all tasks for the kth service processed on ES m during the tth time slot can be calculated as

$$E_m^{k,comp}(t) = \sum_{m'\in\mathcal{M}} N_{m',m}^k(t)k_m f_m^k(t)^2 C_k, \quad (10)$$

where the summation runs over all APs. Therefore, the total computational energy consumption of all tasks for the kth service in the tth time slot can be calculated as

$$E_k^{comp}(t) = \sum_{m\in\mathcal{M}} E_m^{k,comp}(t). \quad (11)$$

Based on the above analyses, the total task processing cost of all requests for the kth service in the tth time slot can be calculated as

$$E_k(t) = E_k^{tran}(t) + E_k^{comp}(t). \quad (12)$$

On one hand, user requests fluctuate over time, meaning the number of requests for the kth service received by each AP changes at every time slot. On the other hand, the distribution of requests for the kth service across different factories is uneven. Therefore, to minimize the average response latency, the task scheduling policy must be dynamically adjusted at each time slot. However, frequent task scheduling leads to high operational costs. Thus, the key question is how to efficiently balance the trade-off between cost and performance.

3 PROBLEM FORMULATION AND ANALYSIS

3.1 Problem Formulation

Considering the fact that enterprises typically operate under the long-term cost budget, we formulate the problem as optimizing task processing performance over the long term within the long-term cost constraint. Specifically, we define E_k^{avg} as the long-term average task processing cost budget for the kth service over T time slots ($T \rightarrow \infty$), which satisfies

$$\lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=1}^{T} E_k(t) \leq E_k^{avg}. \tag{13}$$

Our goal is to minimize the long-term average request response latency, which can be formulated as the following stochastic optimization problem:

$$
\begin{aligned}
\mathcal{P}_1 : &\min_{\mathcal{N}_k} \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=1}^{T} T_k(t) \\
&\text{s.t. } (a) : (1), (2), (7), (12), (13), \\
&\quad (b) : N_{m,m'}^k(t) \in \mathbb{N}, \quad \forall m, m' \in \mathcal{M},
\end{aligned}
\tag{14}
$$

where $\mathbb{N}$ denotes the set of non-negative integers, $\mathcal{N}_k = \{\mathcal{N}_k(t)|t = 1, 2, \cdots, T\}$, and $\mathcal{N}_k(t) = \{N_{m,m'}^k(t)|m, m' \in \mathcal{M}\}$.

The primary challenge in addressing problem $\mathcal{P}_1$ lies in the requirement for future system information (i.e., the evolution of request distribution and resource allocation for the kth service over time) to enable the decision controller to make globally optimal task scheduling decisions in each time slot. Unfortunately, prediction-based methods often fail to accurately capture the state of a time-varying system, whereas queuing-theory-based methods depend on a priori information (e.g., task arrival and service rates) and assume specific distributional patterns for request arrivals and departures, which are frequently difficult to satisfy. Fortunately, in queuing theory, the long-term task processing cost budget constraint (13) in this optimization problem can be interpreted as a form of queue stability control. Moreover, the Lyapunov optimization technique provides an efficient approach to decouple the long-term problem. It does not require any prior system information and can maintain queue stability in an online manner. Therefore, we employ Lyapunov optimization theory to transform the original problem into a series of real-time minimization problems, leading to the development of an online task scheduling algorithm.

3.2 Graph Model and Optimal Solution Analysis

To enhance the understanding of the problem, we model the system state and task scheduling policy using time-varying graph models, which are described separately below.

3.2.1 System State Graph Model

Based on the system model in Section 2 and the problem description in the previous subsection, the system state of the kth service can be represented as a time-varying complete undirected graph, denoted by $\mathcal{G}_t^k = \{\mathcal{V}_t^k, \mathcal{E}_t^k\}$, where:
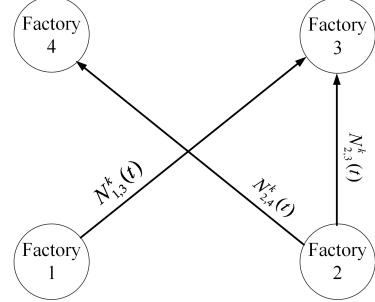

Fig. 4. An example of a weighted directed graph for the task scheduling decision of the kth service at the tth time slot. This example shows that, in the current time slot, the AP of factory 2 will send $N_{2,3}^k(t)$ requests for app_k to the ES of factory 3 and $N_{2,4}^k(t)$ requests to the ES of factory 4. The AP of factory 1 will send $N_{1,3}^k(t)$ requests for app_k to the ES of factory 3.

- $\mathcal{V}_t^k$ denotes the set of vertices representing the edge nodes. Each vertex $m \in \mathcal{V}_t^k$ is associated with two attributes: $N_m^k(t)$ and $F_m^k(t)$, as illustrated in Fig. 2.
- $\mathcal{E}_t^k$ denotes the set of edges, defined as $\mathcal{E}_t^k = \{e_{m,m'}^k \mid m, m' \in \mathcal{V}_t^k\}$. Each edge $e_{m,m'}^k \in \mathcal{E}_t^k$ is associated with two attributes: $R_{m,m'}^k(t)$ and $E_{m,m'}^k$.
- The graph $\mathcal{G}_t^k$ also possesses two global properties: E_k^{avg} and $\boldsymbol{E}_k^{t-} = \{E_k(l) \mid l = 1, 2, \ldots, t-1\}$, where $\boldsymbol{E}_k^{t-}$ represents the known past task processing cost of the kth service up to the tth time slot.

3.2.2 Task Scheduling Policy Graph Model

Based on the system model in Section 2, the variables to be optimized in $\mathcal{P}_1$ can be modeled as a time-varying weighted directed graph. Fig. 4 illustrates an example of a weighted directed graph for the task scheduling decision of the kth service at the tth time slot. Each directed edge represents cross-factory task scheduling, and the weight corresponds to the number of forwarded requests. Each node corresponds to a factory or an edge node.

3.2.3 Optimal Solution Analysis

Based on the above description, the following theorem can be stated:

Theorem 1. *For the kth service, the optimal task scheduling decision in any time slot forms a weighted directed acyclic graph that satisfies the following conditions:*

1) *The graph does not contain intermediate nodes, meaning only source nodes, sink nodes, and isolated nodes exist.*
2) *Nodes in set $\{m|F_m^k(t) = 0\}$ are source nodes.*
3) *For any ES m corresponding to a source node, if $F_m^k(t) = 0$, the sum of the weights of all outgoing edges equals $N_m^k(t)$. If $F_m^k(t) > 0$, the sum is less than or equal to $N_m^k(t)$.*
4) *A sink node must exist whenever there is a source node.*

Proof. The detailed proof is given in Appendix A of the supplemental material. ∎

4 ONLINE TASK SCHEDULING ALGORITHM

In this section, we propose a novel framework for making online task scheduling decisions that are aware of spatiotemporal non-uniformity. To solve $\mathcal{P}_1$, we first transform

the original problem into a queue stability control problem using Lyapunov optimization.

4.1 Problem Transformation

4.1.1 Construct a Virtual Queue

A key concept in Lyapunov optimization is to decompose the long-term optimization metric with long-term constraints into per-time-slot optimization. This allows for direct optimization of the objective function in each time slot, while adhering to the necessary constraints of that slot, such as maintaining virtual queue stability. Therefore, for the constraint $lim_{T\to\infty}\frac{1}{T}\sum_{t=1}^{T}E_k(t) \leq E_k^{avg}$, we define a virtual queue $Q_k(t)$ with an initial value of 0:

$$Q_k(t+1) = \begin{cases} 0, & t = 0 \\ max\{Q_k(t) + E_k(t) - E_k^{a\nu g}, 0\}, & t = 1, 2, \dots \end{cases},$$
(15)

where $Q_k(t)$ represents the virtual control queue length for the kth service at the tth time slot. Based on this definition, we present the following theorem.

Theorem 2. *The long-term average task processing cost constraint (13) is satisfied when the virtual queue remains stable, which occurs when* $\lim_{T\to\infty} \mathbb{E}[Q_k(T)]/T = 0.$

Proof. The detailed proof is given in Appendix B of the supplemental material. ∎

4.1.2 Maintain Queue Stability

To stabilize the virtual queue, the Lyapunov function is first defined as:

$$L(Q_k(t)) = \frac{1}{2}Q_k(t)^2.$$
(16)

The above equation represents a scalar measure of virtual queue congestion for task processing cost, where a smaller $L(Q_k(t))$ indicates less virtual queue backlog. Thus, maintaining the Lyapunov function at a bounded level through a policy implies that the virtual queue is stable. To this end, we introduce the Lyapunov drift, defined as:

$$\Delta(Q_k(t)) = \mathbb{E}[L(Q_k(t+1)) - L(Q_k(t))|Q_k(t)].$$
(17)

$\Delta(Q_k(t))$ represents the increase in virtual queue backlog from time slot t to $t + 1$, which is determined by the task scheduling decision.

4.1.3 Drift-plus-penalty Algorithm

After introducing the virtual queue, the original problem can be transformed into optimizing a new objective function, while ensuring that constraint in the current time slot is maintained. A practical approach is to minimize the Lyapunov drift plus penalty function, i.e., $\min \Delta(Q_k(t)) + VT_k(t)$ in each time slot. This incorporates queue stability into task response latency performance, with the weight V balancing the trade-off between the two. Furthermore, the following theorem provides performance guarantees for this drift-plus-penalty function:

Theorem 3. *For* $\Delta(Q_k(t))$*, generated by applying any task scheduling policy in each time slot, the following equation holds:*

$$\Delta(Q_k(t)) + VT_k(t) \leq B + VT_k(t) + Q_k(t)(E_k(t) - E_k^{avg}),$$
(18)

Algorithm 1: Drift-plus-penalty Algorithm

1 At the beginning of the tth time slot, obtain the distribution of request arrivals, resource allocations, and the value of the virtual queue for service k;
2 Determine the optimal task scheduling decision $\mathcal{N}_k(t)^*$ by solving problem $\mathcal{P}_2$;
3 Perform task scheduling based on $\mathcal{N}_k(t)^*$;
4 Calculate $Q_k(t + 1)$ according to Eq.(15);
5 $t \leftarrow t + 1$;
6 **goto** line 1;

---

where $B = \frac{1}{2}E_k^{\max 2}$ *is a constant across all time slots, and* $E_k^{max} = \max E_k(t) - E_k^{avg}$.

Proof. The detailed proof is given in Appendix C of the supplemental material. ∎

According to Theorem 3, the original problem can be solved by minimizing the right-hand side of (18) at each time slot, transforming the problem into $\mathcal{P}_2$.

$$\mathcal{P}_2 : \min_{\mathcal{N}_k(t)} B + VT_k(t) + Q_k(t)(E_k(t) - E_k^{avg})$$
$$\text{s.t. } (a) : (1), (2), (7), (12), (14b).$$
(19)

This approach eliminates the need for information from future time slots to calculate $\Delta(\Theta_k(t))$. However, it does not yield the optimal solution to the original problem. Later, we will analyze the gap between the solution from this method and the original optimal solution. Algorithm 1 provides the specific form of the drift-plus-penalty algorithm.

4.2 A Heuristic-Based Hierarchical Optimization Approach

In this subsection, we derive the optimal solution for problem $\mathcal{P}_2$ in each time slot. As stated in Theorem 4, it is not feasible to find an optimal solution to problem $\mathcal{P}_2$ within polynomial time [42].

Theorem 4. *The problem* $\mathcal{P}_2$ *is NP-hard.*

Proof. The detailed proof is given in Appendix D of the supplemental material. ∎

To efficiently solve problem $\mathcal{P}_2$, Theorem 1 allows us to classify all factories/regions into three categories under the optimal task scheduling decision: source nodes, sink nodes, and isolated nodes. Once the categories of factories/regions are determined, we only need to decide the number of task requests dispatched from each source node to each sink node, thereby reducing the search space under the constraints of Theorem 1. Next, we first determine the optimal task scheduling decision for a given factory/region category assignment and then solve for the optimal category assignment scheme.

4.2.1 Optimal Task Scheduling for a Given Factory/Region Category Assignment

Given a factory/region category assignment solution, represented by $\mathcal{O}$, $\mathcal{P}$, and $\mathcal{Q}$, where $\mathcal{O}$ indicates isolated nodes, $\mathcal{P}$ represents source nodes, and $\mathcal{Q}$ denotes sink nodes, the

problem then reduces to determining the optimal number of task requests forwarded from each source node to each sink node. This solution should satisfy all constraints while minimizing the objective function $\mathcal{P}_2$. At this stage, $\mathcal{P}_2$ becomes

$$\mathcal{P}_3 : \min_{\{N_{p,q}^k(t)|p\in\mathcal{P}, q\in\mathcal{Q}\}} B + VT_k(t) + Q_k(t)\Big(E_k(t) - E_k^{avg}\Big)$$

$$\text{s.t. } (a): \sum_{q\in\mathcal{Q}} N_{p,q}^k(t) \leq N_p^k(t), \forall p \in \mathcal{P}, F_p^k(t) > 0$$

$$(b): \sum_{q\in\mathcal{Q}} N_{p,q}^k(t) = N_p^k(t), \forall p \in \mathcal{P}, F_p^k(t) = 0$$

$$(c): N_{p,q}^k(t) \in \mathbb{N}, \forall p \in \mathcal{P}, \forall q \in \mathcal{Q}$$

$$(d): (7), (12)$$

$$(20)$$

$\mathcal{P}_3$ is an integer nonlinear programming (INLP) problem, for which we can apply the harmony search (HS) algorithm. As an intelligent optimization method, HS simulates a musician's improvisation process to find the optimal strategy for achieving harmony [43]. To apply the HS algorithm to the constrained optimization problem $\mathcal{P}_3$, we introduce a penalty function, transforming $\mathcal{P}_3$ into $\mathcal{P}_4$:

$$\mathcal{P}_4 : \min_{\{N_{p,q}^k(t)|p\in\mathcal{P}, q\in\mathcal{Q}\}} f(\{N_{p,q}^k(t)|p\in\mathcal{P}, q\in\mathcal{Q}\})$$

$$\text{s.t. } (a): (7), (12), (20c)$$

$$(21)$$

where $f(\{N_{p,q}^k(t)|p \in \mathcal{P}, q \in \mathcal{Q}\}) = B + VT_k(t) + Q_k(t)(E_k(t) - E_k^{avg}) + \frac{1}{\mu}(\sum_{p\in\mathcal{P}} \mathbb{I}(F_q^k(t) > 0)max(0, \sum_{q\in\mathcal{Q}} N_{p,q}^k(t) - N_p^k(t)) + \sum_{p\in P} \mathbb{I}(F_q^k(t) = 0)max(0, |\sum_{q\in\mathcal{Q}} N_{p,q}^k(t) - N_p^k(t)|))$ and $\mu > 0$. $\mathbb{I}(\cdot)$ is the indicator function and equals 1 (resp., 0) if the condition is true (resp., false). As μ approaches 0, the optimal solutions of $\mathcal{P}_4$ and $\mathcal{P}_3$ become increasingly similar.

The HS algorithm mainly involves the following basic parameters:

- *Harmonic memory size (HMS)*: The harmonic memory (HM) is modeled as a $HMS \times |\mathcal{P}||\mathcal{Q}|$ matrix, where each row is a solution to the problem $\mathcal{P}_4$;
- *Number of improvisations (NI)*: The maximum number of iterations to obtain the optimal harmony;
- *Harmonic memory consideration rate (HMCR)*: The probability of taking out a harmony from harmonic memory;
- *Pitch adjusting rate (PAR)*: The probability of adjusting pitch;
- *Step width (SW)*: The adjustment range of pitch;

For illustration, let $sol^j = \{N_{p,q}^{k,j}(t) \mid p \in \mathcal{P}, q \in \mathcal{Q}\}$ represent the jth row of the HM, where $j \in \{1, 2 \ldots, HMS\}$. sol^B and sol^W denote the best and worst solutions in the HM, while $UB(i)$ and $LB(i)$ represent the upper and lower bounds of the ith element, and $sol^j(i)$ represents the ith element of sol^j ($i \in \{1, 2, \cdots, |\mathcal{P}||\mathcal{Q}|\}$). Algorithm 2 summarizes the detailed algorithmic procedure.

Notably, the optimal solution for $\mathcal{P}_4$ may not satisfy the constraints of $\mathcal{P}_3$. Therefore, after Algorithm 2 provides an initial solution, adjustments are necessary to ensure it meets $\mathcal{P}_3$'s constraints. This adjustment process aims to make the final solution compliant with the constraints while maintaining proximity to the initial result. Various adjustment methods are available, though details are omitted here.

Algorithm 2: The Implementation of HS Algorithm

Input: Set parameters: $HMS, NI, SW, HMCR, PAR, UB(i), LB(i)$

Output: The optimal solution of $\mathcal{P}_4$

1 Initialize the harmonic memory based on each element's upper and lower bounds. Substitute each initialized solution into $\mathcal{P}_4$ to calculate the objective function value $f(\cdot)$ and derive sol^B and sol^W;

2 **for** $Itr = 1$ **to** NI **do**

3   Initialize the new solution sol^{new} as an all-zero vector of length $|\mathcal{P}||\mathcal{Q}|$;

4   **for** $i = 1$ **to** $|\mathcal{P}||\mathcal{Q}|$ **do**

5   **if** $rand() < HMCR$ **then**

6   Randomly select a row from the harmonic memory and assign its ith element to $sol^{new}(i)$;

7   **end**

8   **else**

9             $sol^{new}(i) = random.uniform(LB(i), UB(i))$;

10   **end**

11   **if** $rand() < PAR$ **then**

12             $sol^{new}(i) += random.uniform(-1, 1) \times SW$;

13             $sol^{new}(i) = min(max(sol^{new}(i), LB(i)), UB(i))$;

14   **end**

15   **end**

16   **if** $f(sol^{new}) < f(sol^W)$ **then**

17   Remove sol^W from harmony memory and add sol^{new};

18   Update sol^B and sol^W;

19   **end**

20 **end**

21 **return** sol^B

4.2.2 Optimal category assignment scheme

We can now derive the optimal task scheduling policy when the factory/region category solution is given. The next step is to determine the optimal category assignment solution. Based on the conditions outlined in Theorem 1, the optimal task scheduling decision allows for only three types of vertex categories. Therefore, we develop an enhanced discrete particle swarm algorithm to determine the optimal vertex category assignment. In this research, a swarm of N particles moves through an M-dimensional search space at a defined speed. Each particle adjusts its position (i.e., vertex category assignment solution) based on its historical best point and the global best point of all particles in the swarm. The ith particle of the swarm ($i = 1, 2, 3, \cdots, N$) consists of the following three vectors:

- *Current position*: $v_i = (v_{i1}, v_{i2}, \ldots v_{iM})$;
- *Historical optimal position*: $p_i = (p_{i1}, p_{i2}, \ldots p_{iM})$;
- *Velocity*: $v_i = (v_{i1}, v_{i2}, \ldots v_{iM})$.

To improve the global search efficiency and accelerate the process, we impose constraints on each particle, requiring them to meet the conditions outlined in Corollary 1.

Corollary 1. *The particle corresponding to the optimal category assignment should satisfy the following conditions.*

1) *Each element in a particle's position vector can only take values from the set $\mathcal{S} = \{0, 1, 2\}$, representing the corresponding vertex as a source, sink, or isolated node, respectively.*

2) *The elements in a particle's position vector corresponding to the set $\{m|F_m^k(t) = 0\}$ can only take the value 0.*

3) *If a position vector contains an element with the value 0, it must also contain an element with the value 1.*

The three conditions above are derived from Theorem 1.1), 2), and 4), respectively. Additionally, for each particle, the velocity in its mth dimension changes according to the following equation:

$$v_{im} = w_m v_{im} + c_1 \cdot \mathrm{rand}() \cdot (p_{im} - x_{im}) \\ + c_2 \cdot \mathrm{rand}() \cdot (p_{gm} - x_{im}), \tag{22}$$

where $p_g = (p_{g1}, p_{g2}, \ldots p_{gM})$ represents the global optimal position of the swarm. The parameter w_m is the inertia weight factor, while c_1 and c_2 are acceleration constants. These three non-negative parameters balance the trade-off between exploration and exploitation, with $w_m + c_1 + c_2 = 1$.

The equation governing the variation in the mth dimension of each particle's position is shown below:

$$X_{im} = \begin{cases} \mathrm{random}\ (S \setminus \{X_{im}\}), & \text{if } rand() < \frac{1}{1+e^{-v_{im}}} \\ X_{im}, & \text{else} \end{cases}, \tag{23}$$

where $\mathrm{random}\ (S \setminus \{X_{im}\})$ indicates a random selection of an element from the set $S \setminus \{X_{im}\}$.

Algorithm 3 outlines the steps for determining the optimal category assignment solution.

At this point, we have outlined all foundational algorithms within the online task scheduling framework. The overall flow of the proposed framework is illustrated in Fig. 5.

4.3 An Imitation Learning-Based Low-Complexity Approach

Although the heuristic-based hierarchical optimization approach reduces the search space for the optimal solution, determining the optimal factory/region category assignment scheme still increases execution latency, particularly in large systems (e.g., multiple factories). Section 6 compares algorithm execution times and demonstrates that the search for the optimal category assignment scheme significantly contributes to total execution delay. Considering the requirement for low-complexity algorithms in IIoT scenarios, this section proposes an imitation learning-based approach.

Imitation learning is a reinforcement learning concept that enables an intelligent agent to achieve expert-level performance on a given task by mimicking an expert's behavior. The core idea of imitation learning is to enable an intelligent agent to learn an expert's decision-making patterns without requiring an explicit reward function [44]. Imitation learning primarily comprises two methods: behavior cloning (BC) and inverse reinforcement learning (IRL). Behavior cloning is the most straightforward approach, treating imitation

---

Algorithm 3: Category Assignment Based on an Enhanced Discrete Particle Swarm Algorithm

---

Input: Particle swarm size N, number of factories M, inertia weight w_m, acceleration constants c_1, c_2, maximum number of iterations I_{max}

Output: Optimal category assignment solution

1 Randomly initialize the positions and velocities of N particles within the M-dimensional problem space. Each particle must satisfy the conditions in Corollary 1, and start with an individual optimal fitness $pbest_i = 0$ and a global optimal fitness $gbest = 0$;

2 **for** $Itr = 1$ **to** I_{max} **do**

3   **for** $i = 1$ **to** N **do**

4   For the ith particle, the category assignment solution is determined from its position vector. Algorithm 2 is used to derive the optimal scheduling decision for this category assignment solution, yielding the objective function value H of $\mathcal{P}_3$. The particle's fitness is then calculated as $1/H$;

5   **if** $1/H > pbest_i$ **then**

6   Update the particle's historical best position p_i to the current position and $pbest_i = 1/H$;

7   **end**

8   **if** $1/H > gbest$ **then**

9   Update p_g to the current position and $gbest = 1/H$;

10   **end**

11   Update ith particle's velocity and position according to equations (22) and (23);

12   If the modified particle does not meet the conditions outlined in Corollary 1, revert to the previous step and re-execute the process until it does;

13   **end**

14 **end**

15 **return** p_g

---

learning as a supervised learning problem in which an intelligent agent is trained on an expert's state-action data pairs to make similar decisions in the same state [45]. Inspired by this, we can model the decisions of the heuristic-based hierarchical optimization approach as expert behaviors and represent the intelligent agent using a deep neural network.

First, we need to specify the state $\mathcal{S}_t$ and action $\mathcal{A}_t$. Based on Section 3, we define the system state graph as the state, merging its two global attributes into $Q_k(t)$ according to Equation (15). The action is a $|\mathcal{M}|$-dimensional vector, where each element takes a value from $\{0, 1, 2\}$, representing a vertex as a source, sink, or isolated node, respectively. We do not directly use the task scheduling decision graph as the action due to its complex constraints (as shown in Theorem 1), which require more data and incur higher training costs to achieve acceptable accuracy. Experimental results indicate that the key factor affecting the algorithm's running time is determining the optimal category assignment scheme. Therefore, we train the intelligent
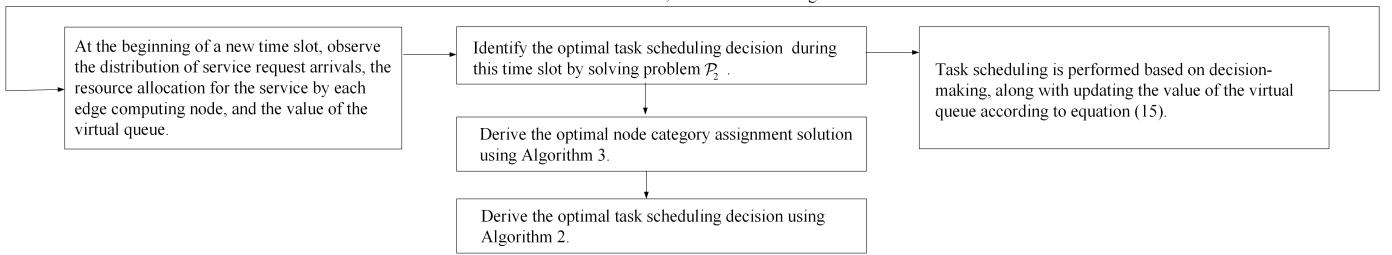
The current time slot ends, and the next one begins.

| At the beginning of a new time slot, observe the distribution of service request arrivals, the resource allocation for the service by each edge computing node, and the value of the virtual queue. | → | Identify the optimal task scheduling decision during this time slot by solving problem $\mathcal{P}_2$. | → | Task scheduling is performed based on decision-making, along with updating the value of the virtual queue according to equation (15). |

Derive the optimal node category assignment solution using Algorithm 3.

Derive the optimal task scheduling decision using Algorithm 2.

Fig. 5. Online task scheduling framework.

agent to learn this determination and subsequently solve the corresponding task-scheduling decision using Algorithm 2.

Specifically, the scheme follows these steps:

- *Step 1*: During the specified period, collect the expert's state-action pairs, denoted as $\{\overline{\mathcal{S}}_t, \overline{\mathcal{A}}_t | t \in \mathcal{I}\}$, where $\overline{\mathcal{A}}_t$ represents the category assignment decision made by the heuristic-based hierarchical optimization approach.
- *Step 2*: Construct a deep neural network as an intelligent agent, consisting of multiple graph convolutional layers and M multilayer perceptrons (MLPs), with the system state graph as input. First, the message-passing mechanism of the graph convolutional layers is used to obtain the embedding representation of each vertex. The embedding representation of the mth vertex is then fed into the mth MLP, whose output dimension is 3, corresponding to the number of vertex categories. Finally, a softmax activation function is applied to obtain the probability of each vertex belonging to different categories.
- *Step 3*: A stochastic gradient descent optimization algorithm (e.g., Adam) is employed to train the intelligent agent constructed in Step 2 using the expert dataset $\{\overline{\mathcal{S}}_t, \overline{\mathcal{A}}_t | t \in \mathcal{I}\}$. The training process minimizes a cross-entropy loss function: $\mathcal{L} = -\sum_{m \in \mathcal{M}} \log(\hat{y}_m)$, where $\hat{y}_m$ denotes the probability predicted by the agent that vertex m belongs to its true label category.
- *Step 4*: Save the trained model parameters, deploy the model to the central controller, and load the trained parameters.
- *Step 5*: During deployment, the intelligent agent predicts the category $\mathcal{A}_t$ of all nodes based on the current system state graph $\mathcal{S}_t$. Given the predicted category assignment scheme, Algorithm 2 is executed to determine the task scheduling decision.

5 THEORETICAL ANALYSIS

5.1 Computational Complexity

5.1.1 Heuristic-Based Hierarchical Optimization Approach

As illustrated in Fig. 5, the task scheduling decision generation process in each time slot involves a two-stage heuristic algorithm. The outer layer addresses the optimal category assignment using an enhanced particle swarm algorithm, while the inner layer determines the optimal scheduling decision for a given category assignment via the harmonic

search algorithm. In Algorithm 3, the outer layer iterates up to I_{max} times, traversing $O(N)$ node category assignment solutions in each iteration, where adjusting each category assignment solution requires traversing M dimensions. Additionally, each category assignment solution requires the execution of Algorithm 2 to compute its corresponding fitness.

In Algorithm 2, up to NI iterations are performed. During each iteration, a harmony is generated, requiring $O(M^2)$ traversals. Therefore, the computational complexity of Algorithm 2 is $O(NI \times M^2)$.

Based on the above analyses, the computational complexity of generating task scheduling decisions in each time slot is $O(I_{max} \times N \times (M + NI \times M^2))$, where I_{max}, NI, and N are algorithm hyperparameters.

5.1.2 Imitation Learning-Based Scheme

In this scheme, the deep neural network first predicts the category assignment scheme, and then Algorithm 2 generates the task scheduling decision based on the given category assignment scheme. In the deep neural network, the node embedding representation is obtained using graph convolutional layers with a complexity of $O(H \times M^2)$, where H is the hidden layer dimension. Each node category prediction is performed by a two-layer MLP with a complexity of $O(H^2)$.

Considering the computational complexity of Algorithm 2, the overall complexity of the scheme can be derived as $O(H \times M^2 + H^2 + NI \times M^2)$. This demonstrates that the proposed scheme lowers the overall computational complexity by reducing the number of executions of Algorithm 2.

5.2 Convergence Performance

Using the proposed algorithms, the near-optimal solution to problem $\mathcal{P}_2$ can be obtained. The convergence of problem $\mathcal{P}_1$ is guaranteed under our online algorithm. Assuming problem $\mathcal{P}_1$ is a bounded function with upper and lower limits T_k^{max} and T_k^{min}, respectively. We define $\tilde{T}_k(t)$ as the delay performance of the proposed algorithm in time slot t, and T_k^{opt} as the lower bound of the time-averaged delay performance with global information. The following theorem provides an upper bound on the time-averaged delay performance and the task scheduling cost queue backlog.

Theorem 5. *For any non-negative control parameter V, the long-term delay performance achieved by the online algorithm satisfies*

the following condition:

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{\tilde{T}_k(t)\} \le T_k^{opt} + \frac{B+D}{V}, \qquad (24)$$

where D represents the upper bound of the performance gap between the solution of problem $\mathcal{P}_2$ obtained using our proposed algorithms and the optimal solution of problem $\mathcal{P}_1$ under global information in each time slot.

Theorem 6. *Given $E_k^{avg} > 0$ and that the initial task processing cost virtual queue backlog is 0, we have the following upper bound for the virtual queue backlog:*

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{Q(t)\} \le \frac{B + V(T^{max} - T^{min}) + D}{\varepsilon}, \quad (25)$$

where $\varepsilon > 0$ is a bounded constant representing the difference between the time-averaged task processing cost under a specific control strategy and the long-term cost budget.

Due to space constraints, the proof process is not elaborated upon here; however, the detailed proof is analogous to that in [46]. According to Theorem 4, the delay performance of the online algorithm gradually approaches the offline optimal solution as the controllable parameter V increases. Additionally, as shown in Theorem 5, the upper bound of the virtual queue backlog for task processing cost is also determined by the parameter V. In summary, there is a performance-cost trade-off of $[O(1/V), O(V)]$ for the online algorithm. Therefore, we can adjust the parameter V to achieve a balance between long-term delay performance and the operational cost.

6 PERFORMANCE EVALUATION

6.1 Simulation Setting

In the simulation, we consider the scenario shown in Fig. 1, where the number of cells is set to 5. According to Section 2, simulating a single service is sufficient; thus, we only consider service k. The computational resources allocated to this service are spatially non-uniform. The number of requests arriving at each AP is spatio-temporally non-uniform. Additionally, the transmission bandwidth allocated to service k between any two factories is between 10 and 100 Gbps, the service request size is between 200 and 500 Kbits, the CPU cycles required to process each request range from 50 to 500 Kilocycles, and the long-term average task processing cost budget is set to 20 J per time slot. The main parameters of the simulation are summarized in Table 3, with all parameters using the values provided unless otherwise specified. All simulations were executed on a Pytorch 1.10.2 platform with an Intel Core i7-13650HX 4.9 GHz CPU and 16 GB of RAM.

To evaluate the performance of the proposed schemes, we compare them against the following benchmark schemes.

1) *Myopic* [47]: The Myopic scheme ignores the queue backlog and minimizes the average processing de-

TABLE 3
Main Simulation Parameters

Parameters	Value
The computational resources allocated by ES 1 to service k $F_1^k(t)$	[50, 60] GHz
The computational resources allocated by ES 2 to service k $F_2^k(t)$	[40, 50] GHz
The computational resources allocated by ES 3 to service k $F_3^k(t)$	[30, 40] GHz
The computational resources allocated by ES 4 to service k $F_4^k(t)$	[20, 30] GHz
The computational resources allocated by ES 5 to service k $F_5^k(t)$	[10, 20] GHz
The number of requests arriving at AP 1 at the start of a time slot for service k $N_1^k(t)$	[10, 20]
The number of requests arriving at AP 2 at the start of a time slot for service k $N_2^k(t)$	[20, 30]
The number of requests arriving at AP 3 at the start of a time slot for service k $N_3^k(t)$	[30, 40]
The number of requests arriving at AP 4 at the start of a time slot for service k $N_4^k(t)$	[40, 50]
The number of requests arriving at AP 5 at the start of a time slot for service k $N_5^k(t)$	[50, 60]
the Cost of scheduling a single request for service k from AP m to ES m' $E_{m,m'}^k(t)$	[0.4, 2.4]
V	10^8
Total number of time slots T	1000
Computational energy efficiency factor k_m	10^{-26}

lay of all requests for service k in each time slot by solving:

$$\mathcal{P}_4 : \min_{\mathcal{N}_k(t)} \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} T_k(t)$$
$$\text{s.t. } (a) : (1), (2), (7), (12), \qquad (26)$$
$$(b) : E_k(t) \le tE_k^{avg} - \sum_{l=1}^{t-1} E_k(l)$$

Here, constraint (b) ensures that the long-term average task processing budget constraint of $\mathcal{P}_1$ is met by the t-th time slot, where $\{E_k(l) \mid l = 1, 2, \cdots, t-1\}$ represents the known past task processing cost up to the t-th time slot.

2) *K time-step-greedy algorithm (KG)* [48]: Task scheduling aimed at minimizing latency is performed every K time slots, without accounting for energy consumption constraints.

3) *No scheduling (NS)*: Tasks are processed immediately upon arrival at the edge computing node, without any task scheduling.

4) *Load-balanced scheduling (LBS)* [49]: Tasks arriving at each edge node are scheduled proportionally to the computational resources allocated to the service by each edge computing node.

5) *Standard heuristic algorithm (SH)*: The transformed problem $\mathcal{P}_2$ is directly solved using a heuristic algorithm, such as the genetic algorithm [15]. The optimization variable is defined as the number of tasks each AP forwards to each ES.

6) *Deep reinforcement learning algorithm (DRL)*: The proximal policy optimization (PPO) algorithm [50] is employed, where the state is defined as the system state graph. The action is defined as the proportion
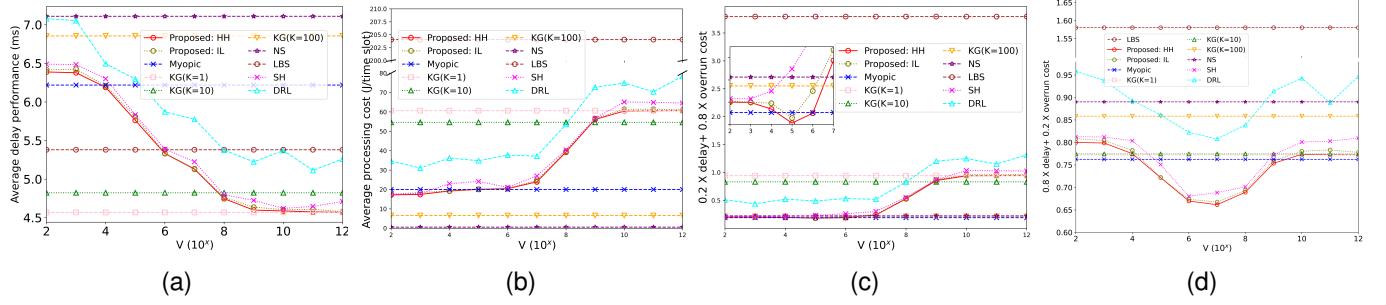
Fig. 6. Performance comparison of different schemes: (a) Achievable delay performance under different schemes. (b) Achievable average task processing cost under different schemes. (c) Algorithm performance when the normalized delay and average overrun cost weights are 0.2 and 0.8, respectively. (d) Algorithm performance when the normalized delay and average overrun cost weights are 0.8 and 0.2, respectively. Note: The normalization of delay and average overrun cost is based on the maximum delay and highest overrun cost observed in the proposed scheme using the heuristic-based hierarchical optimization approach. The legends "Proposed: HH" and "Proposed: IL" represent the online task scheduling scheme based on heuristic hierarchical optimization and the online task scheduling scheme based on imitation learning, respectively.

of tasks each AP forwards to each ES. The reward function is given by

$$R(t) = -(VT_k(t) + max\{(E_k(t) - E_k^{avg}), 0\}). \quad (27)$$

6.2 Simulation Results

The detailed results of our simulation are provided in this section. We compare the effectiveness of the proposed schemes with several above-mentioned benchmark policies, and analyze the performance of the online task scheduling scheme based on heuristic hierarchical optimization under various algorithm configurations and environmental settings.

6.2.1 Algorithm Performance Comparison

Fig. 6 presents the performance of various schemes in terms of delay, average task processing cost, and the weighted sum of the normalized delay and average overrun cost. First, it can be seen that the performance of the two proposed schemes is nearly identical. Fig. 6(a) and (b) show that as V increases, the average task processing delay of the two proposed schemes decreases, whereas the long-term average task processing cost gradually rises. Additionally, the delay and average task processing cost of the two proposed schemes remain stable when V is either very small or very large, corresponding to prioritizing task processing cost and delay, respectively. Both proposed schemes satisfy the long-term average task processing cost constraint when $V < 10^6$. For $10^4 < V < 10^6$, the two proposed schemes achieve optimal delay performance under this constraint. However, LBS and KG ($K = 1, 10$) algorithms fail to meet the long-term energy constraint despite providing better delay performance. As K increases, the average task processing delay under the KG algorithm gradually rises, whereas the corresponding energy consumption decreases. Therefore, in the KG algorithm, K has a similar effect to V in the proposed schemes. Adjusting K enables a trade-off between task processing delay and cost. However, the KG algorithm exhibits significant performance fluctuations over time slots, resulting in higher user experience variability. In contrast, the proposed schemes can provide a more stable user experience.

Fig. 6(c) and (d) show the weighted sum of normalized delay and average overrun cost. In Fig. 6(c), the objective function assigns a weight of 0.2 to delay and 0.8 to overrun cost, while in Fig. 6(d), these weights are 0.8 and 0.2, respectively. As shown in Fig. 6(c), the two proposed schemes achieve optimal performance for the given metric at $V = 10^5$. In Fig. 6(d), the two proposed schemes achieve optimal performance for the given metric at $V = 10^7$. Therefore, the proposed schemes can be tuned to achieve optimal performance by adjusting the value of V according to the weighting preferences for overrun cost and delay.

Additionally, the SH and DRL schemes consistently perform worse than the two proposed schemes, with the DRL scheme exhibiting significant volatility. This is because the SH scheme contains numerous redundant or invalid solutions in its search space, reducing optimization efficiency. In contrast, the DRL scheme suffers from low sample efficiency and an unstable training process, causing early-stage strategies to be random and volatile.

6.2.2 Scalability

Fig. 7 compares the scalability of the two proposed schemes against the SH and DRL algorithms across multiple plants. The number of factories and the long-term average task processing cost budget are scaled by multiplying the default parameters by factors from 1 to 6. The resource allocation and request arrival settings for every additional five factories remain the same as those for the initial five factories. Fig. 7 (a) presents the delay performance of different schemes. The average delay of the two proposed schemes and the SH algorithm gradually decreases as the number of factories increases. This is primarily because a higher number of factories enhances inter-factory collaboration, thereby reducing the average task processing delay. In contrast, the average delay of the DRL algorithm increases at larger M as the number of factories grows. This is because the action space of the task scheduling policy expands quadratically with the number of factories, making it harder for the DRL algorithm to converge. At this stage, the DRL-generated task scheduling policy is largely random, leading to degraded delay performance. Fig. 7 (b) presents the long-term average energy consumption of different schemes. The DRL scheme consistently underperforms compared to the Lyapunov-
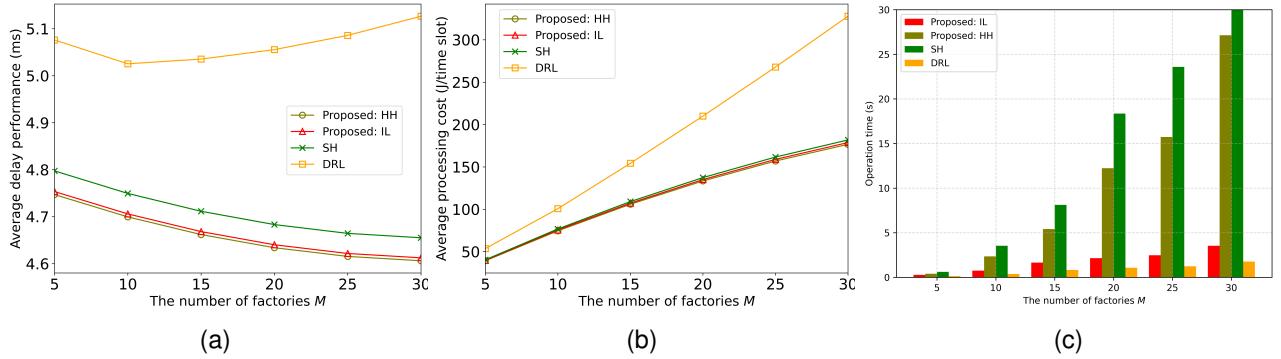
Fig. 7. Scalability comparison of different schemes: (a) Achievable delay performance variation with the number of factories. (b) Average task processing cost variation with the number of factories. (c) Average algorithm running time variation with the number of factories. Note: In Fig. 7 (a) and (b), the number of iterations for the SH algorithm is set to match the total iterations of the HH algorithm. In Fig. 7 (c), the stopping condition for the SH algorithm is defined as reaching a target value as close as possible to that of the HH algorithm.

Fig. 8. Adaptive capability of the proposed scheme to the time-varying system state. (a) Variation of task processing cost over time for different values of V. (b) Variation of the virtual queue backlog over time for different values of V. (c) Distribution of task processing delays in each time slot for different values of V.

based scheme, primarily due to its low sample efficiency. It requires a sufficient number of samples before optimizing its strategy, which initially behaves as a random strategy. Additionally, the DRL scheme's performance heavily depends on the reward function design. Even a slight modification can cause convergence to a completely different strategy.

Fig. 7(c) presents the average running time of different algorithms versus the number of factories M. When the number of factories is small (e.g., $M = 5$), all algorithms exhibit similar average running times. However, as M increases, the running times of the proposed HH algorithm and the SH algorithm grow significantly. Moreover, the proposed HH algorithm consistently exhibits a shorter running time than the SH algorithm. This is because the proposed HH algorithm reduces the search space for the optimal solution through node category assignment. While the running time of the proposed IL scheme increases with M, its growth is less significant than that of the proposed HH algorithm. This indicates that the search process for optimal category assignment is the primary factor contributing to the increased running time of the proposed HH algorithm. Additionally, although the DRL algorithm consistently exhibits the shortest running time, its delay and cost performance remain the worst, with a significant gap compared to the other three schemes. According to Fig. 6 and Fig. 7, the proposed HH scheme is preferable

for smaller networks, and the proposed IL scheme is better suited for large-scale networks. This conclusion aligns with our theoretical analysis of the computational complexity of the two schemes.

6.2.3 Adaptability to System State Changes

Fig. 8(a) shows the task processing cost in each time slot. At smaller values of V, the task processing cost rapidly converges to the expected long-term average threshold. As V increases, the convergence of the task processing cost curve gradually slows. For $V > 10^8$, the task processing cost curve does not converge to the expected long-term average threshold. As V increases further, the convergence value rises, convergence slows, and task processing cost fluctuations become more pronounced. This occurs because increasing V reduces the emphasis on controlling energy consumption that exceeds the expected long-term average, weakening task processing cost control and causing the curve to fluctuate significantly or even fail to converge.

Fig. 8(b) shows the variation of the virtual queue backlog over time for different values of V. The virtual queue backlog remains near zero for $V < 10^8$. This occurs because the proposed scheme tightly controls task processing cost when V is small, ensuring that processing energy consumption in each time slot does not exceed the long-term average task processing cost budget. At $V = 10^8$, the virtual queue back-
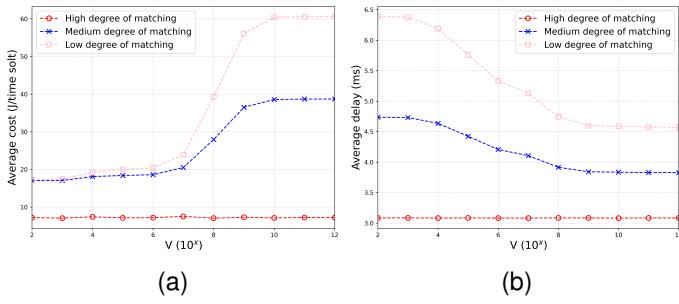
Fig. 9. Impact of the alignment between resource allocation and request distribution on algorithm performance. (a) The long-term average task processing energy consumption with different values of V. (b) The average response delay with different values of V.
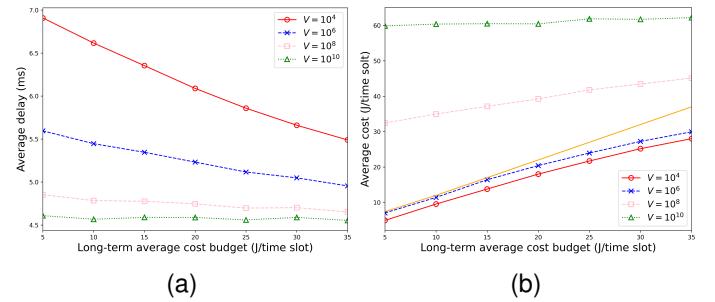

Fig. 10. Impact of long-term average cost constraint on algorithm performance. (a) Variation in average response latency with the long-term average cost constraint. (b) Variation in average processing energy consumption with the long-term average cost constraint.

log initially grows before stabilizing. This corresponds to Fig. 8(a), where the task processing cost gradually converges from a higher value to the long-term average task processing cost budget, after which the virtual queue backlog stabilizes. For $V > 10^8$, the virtual queue backlog does not converge and grows rapidly as V increases, corresponding to the behavior in Fig. 8(a).

Fig. 8(c) shows the distribution of task processing delays across time slots for different values of V. As V increases, the median task processing delay decreases. Additionally, for $V > 10^4$, the distribution of task processing delays becomes more compact as V increases. This occurs because as V increases, the algorithm places more emphasis on minimizing delay, leading to tighter control over delay fluctuations across time slots.

6.2.4 Impact of Resource Allocation and Request Distribution Match Degree on Algorithm Performance

Fig. 9 illustrates the impact of the degree of match between resource allocation and request distribution on algorithm performance. For high matching, the ranges of values for $F_1^1(t)$ to $F_1^5(t)$ are [10, 20], [20, 30], [30, 40], [40, 50], and [50,60]; for medium matching, the ranges of values are [50, 60], [20, 30], [30, 40], [40, 50], and [10, 20]; for low matching, the ranges of values are [50, 60], [40, 50], [30, 40], [20, 30], and [10, 20], respectively. As shown in Fig. 9(a) and (b), a low alignment between request distribution and resource allocation will lead to increased latency and higher task processing cost. The difference in processing cost becomes more significant as V increases, while the impact of matching on average response delay is more pronounced at smaller values of V. Based on the above analysis, the degree of matching between resource allocation and task request distribution determines the upper bound of scheduling policy performance. As preliminary steps, service deployment and resource allocation play a dominant role in determining scheduling policy performance. Task scheduling aims to optimize long-term service quality for spatiotemporally varying requests, subject to a long-term average cost constraint, once service deployment and resource allocation are complete.

6.2.5 Impact of the Long-Term Average Cost Budget on Algorithm Performance

Fig. 10 illustrates the variation in delay and cost performance of the proposed scheme under different values of V and the long-term average cost constraint. At very large values of V, both delay and cost remain nearly constant. For other values of V, as the long-term average cost constraint is relaxed, the proposed scheme achieves a lower average task processing delay while average task processing energy consumption increases gradually. This occurs because relaxing the long-term average energy consumption constraint increases the cost budget for task processing, allowing the more optimal solution to be found over a larger search space. The orange solid line in Fig. 10(b) represents the long-term average energy consumption expectation. It can be seen that the proposed algorithm minimizes delay performance only when $V < 10^6$ while strictly ensuring that the long-term average energy consumption constraint is not exceeded.
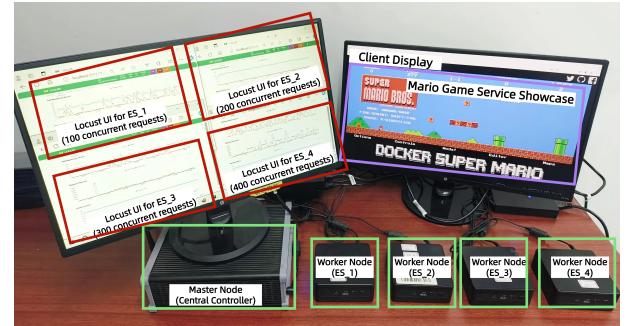
6.3 Practicability Evaluation

Fig. 11. Testbed for our proposed task scheduling scheme.

In this section, we assess the practicality of the proposed scheme. As shown in Fig. 11, we build a Kubernetes-based testbed, consisting of four Intel minicomputer hosts as worker nodes (representing the four ESs integrated with the APs in Fig. 1) and one GeeFlex host as the master node (corresponding to the central controller). Each worker node deploys a gateway (corresponding to the AP in Fig. 1) and an application pod (Mario's cloud gaming service). We allocate 2 CPU cores and 2 GB of RAM to each service pod by configuring the Kubernetes resource file. Using Istio, an open-source service mesh, we configure the gateway of each worker node to distribute incoming user requests to the application pods based on predefined weights [51]. Our algorithm is used to adjust the weight (scheduling
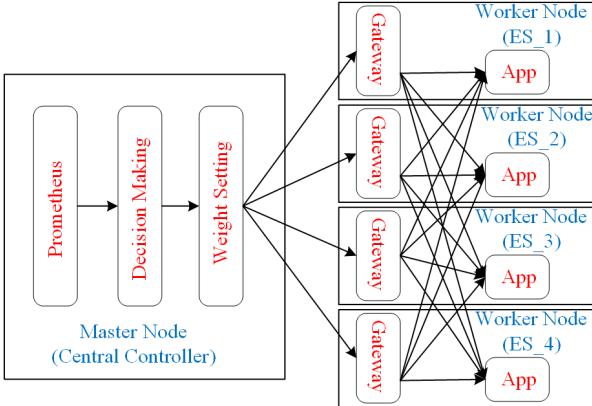
Fig. 12. System Architecture.

TABLE 4
Average Response Latency Comparision of the Five Algorithms

Scheme	Feature	Latency
Proposed Scheme	Introduce a graph model to guide optimal task scheduling decisions	33 ms
Round-robin Scheduling	Distribute requests to available server nodes sequentially and cyclically.	49 ms
Load-based Scheduling	Distribute requests according to each server's current load.	41 ms
Least-connection Scheduling	Prioritize scheduling requests to servers with the fewest active connections.	43 ms
Direct Processing	Each edge node directly handles incoming requests without scheduling.	62 ms

probability) of each gateway to each service pod, achieving task scheduling across nodes. As shown in Fig. 12, the algorithm module is deployed on the master node. The Prometheus module monitors the system state, while the Decision Making module periodically reads the system state from the Prometheus module and runs the algorithm (Due to practical difficulties in measuring energy consumption from task scheduling, we simplify the evaluation by considering only response latency.). The decision result, i.e., the weight of each gateway to each service pod, is generated and transmitted to the Weight Setting module. The Weight Setting module modifies the weight of each gateway to each service pod based on the decision from the Decision Making module.

To validate the performance of our scheme, we use Locust, a Python-based performance testing framework, to send varying numbers of concurrent requests to each worker node's gateway, simulating spatial non-uniformity in user request distribution. In our experiments, the numbers of concurrent requests sent to the four gateways are set to 100, 200, 300, and 400, with the average response latency of all requests as the observation metric.

We compare our proposed scheme with task scheduling strategies frequently used in engineering practice: round-robin scheduling, load-based scheduling, least-connection scheduling, and direct processing. The comparison result in Table 4 shows that our proposed scheme achieves lower task response latency. This is primarily because our scheme unifies the consideration of both computing power and network states. The average task response latency under our scheme is reduced by approximately 20% compared to the load-based task scheduling mechanism.

7 CONCLUSION

In this paper, we address the problem of optimizing the performance of collaborative edge network services under a long-term average task processing cost budget. We implement a spatiotemporal non-uniformity-aware online task scheduling framework to balance average response delay and task scheduling cost. To handle unpredictable future system states, we incorporate the long-term budget into real-time optimization problems using Lyapunov optimization techniques. To address the spatial non-uniformity

of user request distribution, we introduce a graph model and derive the conditions for the optimal solution. We then employ an enhanced discrete particle swarm algorithm and a harmonic search algorithm to get the near-optimal solution based on the derived conditions. To accelerate the algorithm's operation further, an imitation-based approach is introduced. We also provide a theoretical analysis of the algorithm's performance. Finally, we conduct extensive simulations to demonstrate the effectiveness of the proposed algorithms and build a prototype testbed to validate the practicality of the proposed scheme.

REFERENCES

[1] D. Georgakopoulos, P. P. Jayaraman, M. Fazia, M. Villari, and R. Ranjan, "Internet of Things and edge cloud computing roadmap for manufacturing," *IEEE Cloud Comput.*, vol. 3, no. 4, pp. 66–73, Jul. 2016.
[2] H. Guo and J. Liu, "Collaborative computation offloading for multiaccess edge computing over fiber–wireless networks," *IEEE Trans. Veh. Technol.*, vol. 67, no. 5, pp. 4514–4526, May. 2018.
[3] J. Ren, G. Yu, Y. He, and G. Y. Li, "Collaborative cloud and edge computing for latency minimization," *IEEE Trans. Veh. Technol.*, vol. 68, no. 5, pp. 5031–5044, May. 2019.
[4] Z. Hong, W. Chen, H. Huang, S. Guo, and Z. Zheng, "Multi-hop cooperative computation offloading for industrial IoT–edge–cloud computing environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 12, pp. 2759–2774, Dec. 2019.
[5] Y. Zhang, X. Lan, J. Ren, and L. Cai, "Efficient computing resource sharing for mobile edge-cloud computing networks," *IEEE/ACM Trans. Networking*, vol. 28, no. 3, pp. 1227–1240, Jun. 2020.
[6] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, 2017.
[7] H. A. Alameddine, S. Sharafeddine, S. Sebbah, S. Ayoubi, and C. Assi, "Dynamic task offloading and scheduling for low-latency IoT services in multi-access edge computing," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 668–682, Mar. 2019.
[8] Z. Qu, X. Zhang, H. Huang, Y. Li, and W. Wang, "Community and priority-based microservice placement in collaborative vehicular edge computing networks," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Dubai, United Arab Emirates, 2024, pp. 1–6.
[9] Y. Sahni, J. Cao, and L. Yang, "Data-aware task allocation for achieving low latency in collaborative edge computing," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 3512–3524, Apr. 2019.
[10] Y. Li, B. Lei, Z. Li, Z. Qu, X. Zhang, and W. Wang, "Task offloading with multi-cluster collaboration for computing and network convergence," in *Proceedings of the 29th Annual International Conference on Mobile Computing and Networking*, Madrid, Spain, 2023, pp. 1–3.
[11] Y. Sahni, J. Cao, L. Yang, and Y. Ji, "Multi-hop multi-task partial computation offloading in collaborative edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 5, pp. 1133–1145, May. 2021.

[12] Y. Deng, H. Zhang, X. Chen, and Y. Fang, "Multi-hop task routing in vehicle-assisted collaborative edge computing," *IEEE Trans. Veh. Technol.*, vol. 73, no. 2, pp. 2444–2455, Feb. 2024.

[13] J. Lin, Y. Miao, L. Wei, T. Leng, and K.-K. R. Choo, "Efficient secure inference scheme in multiparty settings for industrial internet of things," *IEEE Trans. Ind. Inf.*, vol. 20, no. 10, pp. 11 877–11 886, Oct. 2024.

[14] J. Wang, J. Tang, Z. Xu, Y. Wang, G. Xue, X. Zhang, and D. Yang, "Spatiotemporal modeling and prediction in cellular networks: A big data enabled deep learning approach," in *IEEE INFOCOM*, Atlanta, GA, USA, 2017, pp. 1–9.

[15] Y. Li, X. Zhang, Y. Sun, J. Liu, B. Lei, and W. Wang, "Joint offloading and resource allocation with partial information for multi-user edge computing," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Rio de Janeiro, Brazil, 2022, pp. 1736–1741.

[16] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 1, pp. 856–868, Jan. 2019.

[17] L. Ma, X. Wang, X. Wang, L. Wang, Y. Shi, and M. Huang, "Tcda: Truthful combinatorial double auctions for mobile edge computing in industrial internet of things," *IEEE Trans. Mob. Comput.*, vol. 21, no. 11, pp. 4125–4138, Nov. 2022.

[18] A. Ebrahimzadeh and M. Maier, "Cooperative computation offloading in FiWi enhanced 4G HetNets using self-organizing MEC," *IEEE Trans. Wireless Commun.*, vol. 19, no. 7, pp. 4480–4493, Jul. 2020.

[19] W. Hou, H. Wen, N. Zhang, J. Wu, W. Lei, and R. Zhao, "Incentive-driven task allocation for collaborative edge computing in Industrial Internet of Things," *IEEE Internet Things J.*, vol. 9, no. 1, pp. 706–718, Jan. 2022.

[20] Y. Sun and X. Zhang, "A2C learning for tasks segmentation with cooperative computing in edge computing networks," in *Proc. IEEE Glob. Commun. Conf. (GLOBECOM)*, Rio de Janeiro, Brazil, 2022, pp. 2236–2241.

[21] T. Tao, J. Hou, and A. Nayak, "A GNN-DRL-based collaborative edge computing strategy for partial offloading," in *Proc. IEEE Glob. Commun. Conf. (GLOBECOM)*, Kuala Lumpur, Malaysia, 2023, pp. 3717–3722.

[22] Y. Li, X. Ge, B. Lei, X. Zhang, and W. Wang, "Joint task partitioning and parallel scheduling in device-assisted mobile edge networks," *IEEE Internet Things J.*, vol. 11, no. 8, pp. 14 058–14 075, Apr. 2024.

[23] Z. Liu, Y. Zhao, J. Song, C. Qiu, X. Chen, and X. Wang, "Learn to coordinate for computation offloading and resource allocation in edge computing: A rational-based distributed approach," *IEEE Trans. Network Sci. Eng.*, vol. 9, no. 5, pp. 3136–3151, Oct. 2022.

[24] J. Amendola, L. R. Cenkeramaddi, and A. Jha, "Drone landing and reinforcement learning: State-of-art, challenges and opportunities," *IEEE Open J. Intell. Transp. Syst.*, vol. 5, pp. 520–539, Aug. 2024.

[25] X. Lin, J. Wu, J. Li, W. Yang, and M. Guizani, "Stochastic digital-twin service demand with edge response: An incentive-based congestion control approach," *IEEE Trans. Mob. Comput.*, vol. 22, no. 4, pp. 2402–2416, Apr. 2023.

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

[27] Y. Shi, C. Yi, B. Chen, C. Yang, K. Zhu, and J. Cai, "Joint online optimization of data sampling rate and preprocessing mode for edge–cloud collaboration-enabled industrial iot," *IEEE Internet Things J.*, vol. 9, no. 17, pp. 16 402–16 417, Sep. 2022.

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

[29] Y. Yang, Y. Shi, C. Yi, J. Cai, J. Kang, D. Niyato, and X. Shen, "Dynamic human digital twin deployment at the edge for task execution: A two-timescale accuracy-aware online optimization," *IEEE Trans. Mob. Comput.*, vol. 23, no. 12, pp. 12 262–12 279, Dec. 2024.

[30] Y. Shi, Y. Yang, C. Yi, B. Chen, and J. Cai, "Towards online reliability-enhanced microservice deployment with layer sharing in edge computing," *IEEE Internet Things J.*, vol. 11, no. 13, pp. 23 370–23 383, Jul. 2024.

[31] A. Petrosino, G. Sciddurlo, G. Grieco, A. A. Shah, G. Piro, L. A. Grieco, and G. Boggia, "Dynamic Management of Forwarding Rules in a T-SDN Architecture with Energy and Bandwidth Con-straints," in *Proc. 19th Int. Conf. Ad-Hoc Netw. Wireless (ADHOC-NOW)*, Bari, Italy, 2020, pp. 3–15.

[32] Z. Jahedi and T. Kunz, "Fast and Cost-Efficient Virtualized Network Function Placement Algorithm in Wireless Multi-hop Networks," in *Proc. 19th Int. Conf. Ad-Hoc Netw. Wireless (ADHOC-NOW)*, Bari, Italy, 2020, pp. 23–36.

[33] A. Pratap and S. K. Das, "Stable matching based resource allocation for service provider's revenue maximization in 5g networks," *IEEE Trans. Mob. Comput.*, vol. 21, no. 11, pp. 4094–4110, Nov. 2022.

[34] Y. Chai, K. Gao, G. Zhang, L. Lu, Q. Li, and Y. Zhang, "Joint task offloading, resource allocation and model placement for ai as a service in 6g network," *IEEE Trans. Serv. Comput.*, vol. 17, no. 6, pp. 3830–3843, Dec. 2024.

[35] R. Liu, P. Yang, H. Lv, and W. Li, "Multi-objective multi-factorial evolutionary algorithm for container placement," *IEEE Trans. Cloud Comput.*, vol. 11, no. 2, pp. 1430–1445, June. 2023.

[36] Z. Chen, W. Yi, A. S. Alam, and A. Nallanathan, "Dynamic task software caching-assisted computation offloading for multi-access edge computing," *IEEE Trans. Commun.*, vol. 70, no. 10, pp. 6950–6965, Oct. 2022.

[37] Z. Yu, J. Hu, G. Min, Z. Zhao, W. Miao, and M. S. Hossain, "Mobility-aware proactive edge caching for connected vehicles using federated learning," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 8, pp. 5341–5351, Aug. 2021.

[38] X. Wang, R. Li, C. Wang, X. Li, T. Taleb, and V. C. Leung, "Attention-weighted federated deep reinforcement learning for device-to-device assisted heterogeneous collaborative edge caching," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 1, pp. 154–169, Jan. 2021.

[39] X. Wei, A. M. Rahman, D. Cheng, and Y. Wang, "Joint optimization across timescales: Resource placement and task dispatching in edge clouds," *IEEE Trans. Cloud Comput.*, vol. 11, no. 1, pp. 730–744, Mar. 2023.

[40] X. Ma, A. Zhou, S. Zhang, and S. Wang, "Cooperative service caching and workload scheduling in mobile edge computing," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Toronto, Canada, 2020, pp. 2076–2085.

[41] B. Li, F. Si, W. Zhao, and H. Zhang, "Wireless powered mobile edge computing with NOMA and user cooperation," *IEEE Trans. Veh. Technol.*, vol. 70, no. 2, pp. 1957–1961, Feb. 2021.

[42] E. Malaguti, M. Monaci, P. Paronuzzi, and U. Pferschy, "Integer optimization with penalized fractional values: The Knapsack case," *European Journal of Operational Research*, vol. 273, no. 3, pp. 874–888, 2019.

[43] Q.-K. Pan, P. N. Suganthan, M. F. Tasgetiren, and J. J. Liang, "A self-adaptive global best harmony search algorithm for continuous optimization problems," *Applied Mathematics and Computation*, vol. 216, no. 3, pp. 830–848, 2010.

[44] M. Zare, P. M. Kebria, A. Khosravi, and S. Nahavandi, "A survey of imitation learning: Algorithms, recent developments, and challenges," *IEEE Trans. Cybern.*, vol. 54, no. 12, pp. 7173–7186, Dec. 2024.

[45] A. O. Ly and M. Akhloufi, "Learning to drive by imitation: An overview of deep behavior cloning methods," *IEEE Trans. Intell. Veh.*, vol. 6, no. 2, pp. 195–209, Jun. 2021.

[46] Y. Mao, J. Zhang, S. Song, and K. B. Letaief, "Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems," *IEEE Trans. Wireless Commun.*, vol. 16, no. 9, pp. 5994–6009, Sep. 2017.

[47] L. Huang, S. Bi, and Y.-J. A. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Trans. Mob. Comput.*, vol. 19, no. 11, pp. 2581–2593, Nov. 2020.

[48] T. Ouyang, Z. Zhou, and X. Chen, "Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 10, pp. 2333–2345, Oct. 2018.

[49] H. Liu, Y. Li, and S. Wang, "Request scheduling combined with load balancing in mobile-edge computing," *IEEE Internet Things J.*, vol. 9, no. 21, pp. 20 841–20 852, Nov. 2022.

[50] H. Li, K. Xiong, Y. Lu, W. Chen, P. Fan, and K. B. Letaief, "Collaborative task offloading and resource allocation in small-cell mec: A multi-agent ppo-based scheme," *IEEE Trans. Mob. Comput.*, vol. 24, no. 3, pp. 2346–2359, Nov. 2024.

[51] L. Calcote and Z. Butcher, *Istio: Up and running: Using a service mesh to connect, secure, control, and observe.* O'Reilly Media, 2019.

Yang Li received the B.S. degree in communication engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 2022, where he is currently pursuing the Ph.D. degree with the Key Laboratory of Universal Wireless Communication, School of Information and Communication Engineering.

His research interests include mobile edge networks, Internet of Things, and resource allocation.

Bo Lei received the master's degree in telecommunication engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 2006.

He is now the deputy director of the Network Technology Research Institute of a director of future network research center of China telecom research institute. He currently leads the Future Network Research Center focusing on future network. He is the first author of two technical books and has published more than 30 papers in top journals and international conferences, and filed more than 30 patents. His currently research interests include future network architecture, new network technology, computing power network, and 5G application verification.

Xing Zhang (Senior Member, IEEE) received the Ph.D. degree from Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in 2007.

He is currently a Full Professor with the School of Information and Communications Engineering, BUPT. He has authored or coauthored five technical books and over 300 papers in top journals and international conferences and holds over 80 patents. His research interests are mainly in 5G/6G networks, satellite communications, edge intelligence, and Internet of Things.

Dr. Zhang has received six Best Paper Awards in international conferences. He has served as a General Co Chair for the Third IEEE International Conference on Smart Data (SmartData-2017) and as a TPC co chair/TPC member for a number of major international conferences.

Yukun Sun received the B.S. degree in communication engineering from Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in 2020. She is currently pursuing the Ph.D.degree with the Key Laboratory of Universal Wireless Communication, School of Information and Communication Engineering, BUPT.

Her research interests include computing power network, computing offloading and resource allocation.

Wenbo Wang (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees from Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in 1986, 1989, and 1992, respectively.

He is a former Vice President and a Professor at BUPT, a Fellow of the China Institute of Communications. He is currently the President of the Xiong'an Institute of Aerospace Information, Supervisor of the Chinese Association for Artificial Intelligence, Chairman of the Academic Work Committee of the China Institute of Communications, and Chairman of the Digital Twin and System Simulation Professional Committee of the China Institute of Communications. His current research interests include radio transmission technology, wireless network theory, cognitive communications, and software radio technology.