





Offloading Dependent Tasks in Edge Computing With Unknown System-Side Information

Xingxia Dai , Zhu Xiao , Senior Member, IEEE, Hongbo Jiang , Senior Member, IEEE, Ming Lei, Geyong Min , Member, IEEE, Jiangchuan Liu , Fellow, IEEE, and Schahram Dustdar , Fellow, IEEE

Abstract—We consider the problem of dependent task offloading in edge computing with unknown system-side information (e.g., edge transmission rate and computation resources). In this problem, tasks have complicated dependency relationships and have no prior knowledge of system-side information to assist offloading decision-making. Although existing learning-based approaches can help to address unknown system-side information, the impact of inherent task dependency on such approaches has not been formally explored. To bridge the gap, we first use a breadth-first-search (BFS) method to decouple task dependency, and then leverage the Lyapunov optimization technique to transfer the long-term offloading problem to an online optimization problem. Furthermore, we employ the multi-armed bandit (MAB) theory to develop the online learning-based dependent task offloading algorithm, called OL-DTO. The algorithm can address the unknown system-side information and is augmented with task dependency awareness. We present a rigorous theoretical analysis to evaluate the performance of this algorithm in terms of application delay and UD energy consumption. Our extensive experimental results demonstrate that the OL-DTO algorithm significantly reduces application delay while satisfying the long-term energy budget constraint of the UD.

Index Terms—Unknown system-side information, task dependency, concurrent-enhanced offloading, learning-based offloading, multi-armed bandit theory.

Manuscript received 30 January 2023; revised 9 July 2023; accepted 25 September 2023. Date of publication 29 September 2023; date of current version 13 December 2023. This work was supported in part by the National Natural Science Foundation of China under Grants 62272152, 62372161, U20A20181, and 62102142, in part by the National Key R&D Program of China under Grant 2022YFE0137700, in part by Hunan Provincial Natural Science Foundation under Grants 2020JJ4211, 2020JJ5089, and 2022JJ40878, in part by the Science and Technology Innovation Program of Hunan Province under Grant 2021RC4023, and in part by the Key Research and Development Program of Hunan Province under Grant 2021WK2001. Recommended for acceptance by C.A. Ardagna. (Corresponding authors: Zhu Xiao; Hongbo Jiang.)

Xingxia Dai, Zhu Xiao, and Hongbo Jiang are with the College of Computer Science and Electronic Engineering, Hunan University, Changsha, Hunan 410082, China, and also with the Shenzhen Research Institute, Hunan University, Shenzhen 518055, China (e-mail: xingxdai718@gmail.com; zh-xiao@hnu.edu.cn; hongbojiang2004@gmail.com).

Ming Lei is with Unicom Digital Technology Company, Ltd., Beijing 100033, China (e-mail: minglei@chinaunicom.cn).

Geyong Min is with the Department of Computer Science, College of Engineering, Mathematics, and Physical Sciences, University of Exeter, EX4 4QF Exeter, U.K. (e-mail: G.Min@exeter.ac.uk).

Jiangchuan Liu is with the School of Computing Science, Simon Fraser University, Burnaby, BC V5A 1S6, Canada, and also with Jiangxing Intelligent R&D Department Inc., Nanjing 210000, China (e-mail: jcliu@sfu.ca).

Schahram Dustdar is with TU Wien, 1040 Vienna, Austria (e-mail: dustdar@infosys.tuwien.ac.at).

Digital Object Identifier 10.1109/TSC.2023.3320674

I. INTRODUCTION

EDGE computing pushes plentiful computation resources (e.g., CPU, memory, and storage) from the centralized cloud to the network edge, which enables densely deployed edge servers to provide computation services for computing-intensive applications [1], [2], [3], [4], [5]. The applications, such as augmented reality and face recognition [6], [7], can be partitioned into multiple tasks [8], [9], [10]. On this basis, task offloading with fine-granularity in edge computing has been extensively studied [11], [12], [13], [14], [15], [16], [17], [18], where the tasks are offloaded from the resource-limited user devices (UDs) to powerful edge servers for low application delay and energy consumption.

To better reap the benefits of edge computing, the UD makes offloading decisions based on current edge offloading performance, which can be obtained by the system-side information (e.g., edge transmission rate and computation resources). In practical edge computing scenarios, however, the system-side information is generally unknown to UD [19], [20], [21]. For example, complicated wireless networks impede the acquisition and prediction of edge transmission rate [22]; besides, an edge computing system involves multiple edge service providers, and the system-side information is usually undisclosed to UD across different service providers [23]. Facing the unknown system-side information, several works [24], [25], [26] implement task offloading through continuously learning historical observations of edge offloading performance.

Despite the efforts, the existing learning-based approaches neglect the impact of inherent dependency on task offloading and cannot be applied to dependent task offloading directly. Task dependency refers to logical precedence and data transfer. In dependent task offloading, a UD makes offloading decisions to determine which edge servers should be selected for processing the current tasks, then the outputs from the preceding tasks need to be transferred to the selected edge servers for triggering the current tasks [27]. Based on the data issued by Alibaba [28], more than 75% of tasks exhibit dependency.

However, many prior studies mostly focus on independent task offloading [29], [30], [31], [32], [33]. Such offloading, due to unreasonable processing compositor, may be trapped by unexpected processing delay and large transfer traffic overhead in practice. Considering that, several works study task offloading with sequential dependency [34] or concurrent dependency [35]. Since these studies greatly simplified complicated generalized dependencies, they cannot reflect practical general dependent

traits. Furthermore, few works use directed acyclic graphs (DAGs) to construct general task dependency [8], [36], [37]. Unfortunately, these works consider dependent task offloading under deterministic system-side information, which is hard to implement in the real world where the system-side information is difficult to obtain.

To achieve efficient dependent task offloading in edge computing with the unknown system-side information, there remain three key challenges. First, task dependency causes the requirements of the logical precedence and data transfer among tasks, which highly couples offloading decision-making and leads to NP-hard dependent task offloading. Second, dependent task offloading incurs additional transmission energy consumption for the UD, but the UD's energy consumption needs to follow the long-term energy budget constraint. If the UD consumes massive energy for current task offloading, the remaining energy for offloading subsequent tasks will be less. Such long-term local energy constraint restricts the short-term dependent task offloading decisions. Third, the unknown system-side information demands learning-based offloading, while task dependency incapacitates the existing learning-based works as the works fail to consider the impact of inherent task dependency on learning-based offloading.

In this article, we investigate the extremely compelling but much less studied problem of dependent task offloading in edge computing with the unknown system-side information. Our goal is to achieve the minimum application delay under the long-term energy budget constraints of the UD. To address the above-mentioned challenges, we first adopt a breadth-first-search (BFS) method to decouple task dependency. To further reduce the application delay, we optimize the BFS-based method to categorize dependent tasks as different *start ranks* and *end ranks*. Tasks in the same start rank have identical logical precedence and can be offloaded concurrently. Then, we leverage a Lyapunov optimization technique to construct a virtual energy deficit queue, transferring the long-term dependent task offloading problem to the online optimization problem. On this basis, we propose a multi-armed bandit (MAB)-based learning algorithm for dependent task offloading. The algorithm continuously learns previously observed edge offloading performance for each start rank and maintains the learning efficiency of tasks in the same end rank. In this way, the algorithm enables to address the unknown system-side information while emphasizing task dependency in the learning-based task offloading.

The main contributions of this article are as follows:

- We investigate the dependent task offloading problem with unknown system-side information. This problem accounts for generalized task dependencies and unknown prior knowledge of the system-side information, which is a representative offloading scenario in practical edge computing. Then, we formalize the problem as a decision-making optimization problem, where dependent tasks make offloading decisions to minimize application delay while staying within the long-term UD's energy budget.
- We propose a novel online learning-based dependent task offloading algorithm, called OL-DTO. The algorithm develops a breadth-first-search (BFS) method to

decouple task dependency and then leverages Lyapunov optimization technique to transform the long-term offloading decision-making problem into an online optimization problem. In the online problem, the OL-DTO algorithm utilizes multi-armed bandit (MAB) theory to address unknown system-side information. With OL-DTO, dependent tasks can be effectively offloaded to edge servers without requiring system-side information, thereby achieving low application delay under the long-term UD's energy constraints.

- We conduct a rigorous theoretical analysis of the OL-DTO algorithm and presented an upper bound on its performance. Furthermore, we conduct extensive experiments across multiple offloading scenarios using real-world applications and measurements. The results demonstrate that the proposed algorithm outperforms other algorithms in terms of application delay, local energy consumption, and learning regret, under varying task delays and learning times.

The remainder of this article is organized as follows. Section II presents the related works. Section III presents the system models and problem formulation, followed by the online learning-based algorithm for dependent task offloading in Section IV. In Section V, we present the performance evaluation, followed by the conclusion in Section VI.

II. RELATED WORKS

Task offloading is a central theme in edge computing. Many prior studies [13], [14], [15], [16], [38], [39], [40], [41], [42], [43], [44], [45], [46] have considered when/how to offload computing-intensive tasks from local UDs to powerful edge servers, with the goal of minimizing the task delay, energy consumption and so forth. These works generally rely on deterministic system-side information to make offloading decisions. However, the information is difficult to obtain due to complicated wireless networks and multiple service providers in practical edge computing scenarios.

To address the unknown system-side information, several works have proposed learning-based approaches [19], [29], [47], [48], [49], [50], [51], [52]. The learning-based approaches construct satisfactory offloading strategies through continuous observations, learning and update. Considering unknown server processing speed and cellular data rate, the authors in [19] design a decentralized user-initiated task offloading approach, where users make offloading decisions based on historical observation of edge offloading performance to optimize user rewards in dynamic task placement scenario. In [29], the authors investigate a deep reinforcement learning method to minimize the energy consumption under task delay constraints by optimizing computing resources, power, and subcarrier allocation under an unknown offloading environment. In [47], the authors develop a decentralized calibrated contextual bandit learning algorithm. The algorithm enables users to learn task computational delay and make offloading decisions without peer-to-peer information exchange among micro base stations, aiming at minimizing the long-term average task delay of all users. In [48], the authors

propose the deep reinforcement learning-based algorithm to cope with a dynamic-channel single-user multi-edge-server MEC system, which achieves a near-optimal offloading solution after enough learning by jointly optimizing transmission duration and computation workloads. In [49], the authors take into account time-varying network dynamics and derive deep reinforcement learning-based solutions for edge server selection, cooperative offloading, and handover cost, with the goal of minimizing the computation task delay cost. The authors in [50] investigate an auction-based cluster federated learning approach in a mobile edge computing system, where data owners enable to conduct model training without sending their raw data to third-party servers. In this way, data privacy in an edge computing system can be greatly enhanced. The authors in [51] study a reinforcement learning-based task offloading scheme for edge-enabled sensor networks. This scheme utilizes reinforcement learning to solve task-offloading problems, so as to minimize latency and energy consumption. The authors in [52] design a blockchain-based framework for ensuring secure task offloading in MEC systems, while staying low execution delay and energy consumption. In the framework, a deep reinforcement learning-based algorithm is proposed to find out the close-optimal task offloading decision.

Unfortunately, the learning-based approaches cannot be applied to dependent task offloading directly as the inherent task dependency is neglected in the approaches. Task dependency refers to logical precedence and data transfer among tasks, which can be divided into three categories, i.e., sequential dependency, concurrent dependency, and general dependency [1]. The former two dependencies as a special case, greatly simplify the impact of task dependency on offloading and hence are adopted in many works. For example, the authors in [34], [53] consider sequential task dependency, where one task is assumed to have only one preceding task and one subsequent task; and the authors in [35], [54] consider concurrent task dependency, where tasks are assumed to have no data transfer and can be processed simultaneously. Since real-world applications mostly have complicated general dependency, few works [8], [9], [10], [36], [37] leverage DAGs to construct the task dependency. The authors in [8] propose a dependent task offloading framework for multiple applications in a cloud-edge collaborative system. In this framework, the terminal user aims to improve the user experience quality by adaptively offloading computation tasks with generalized dependency constraints to cloud or edge computing servers for processing. The authors in [36] construct a DAG for all current dependent tasks and propose the optimization goal of minimizing the task deadline violation rate. Based on this optimization goal, they designed a task scheduling algorithm that supports multiple priorities. This algorithm determines the optimal execution order of tasks by introducing multiple task priority indicators to prioritize task collaboration. The authors in [37] design a task dependency model by integrating task offloading decisions and data transmission relationships. Based on the generalized dependency relationship between tasks, they propose an optimization problem with task offloading, drone trajectory, and resource allocation decisions as variables. By decoupling the above optimization problem into two sub-problems

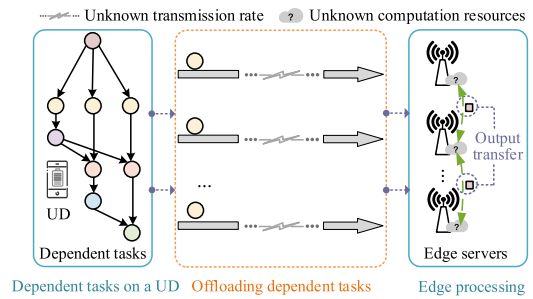


Fig. 1. System illustration. A UD makes offloading decisions to determine which edge servers should be selected for dependent task processing in edge computing with unknown system-side information.

and solving them separately, low-energy dependent task offloading is achieved.

The authors in [9] consider a dependent task offloading problem to minimize the overall application execution cost. The authors in [10] propose an energy-efficient dependent computation offloading problem for the minimum energy consumption. Yet, the above-mentioned works consider task dependency as a constraint in their optimization problems but do not derive a solution to decouple the dependency. Furthermore, these works rely on deterministic system-side information to assist the offloading decision-making and hence are hard to implement in practical edge computing scenarios where the system-side information is difficult to obtain.

Different from the existing studies, this work considers a problem of dependent task offloading in edge computing with unknown system-side information, and we propose innovative solutions to the unique problem. To the best of our knowledge, this problem has not been studied in the existing literature.

III. SYSTEM MODELS AND PROBLEM FORMULATION

We consider an edge computing system that consists of a set of edge servers $\mathcal{M} = \{1, \dots, M\}$ and a UD, as presented in Fig. 1. The UD operates computing-intensive applications, and each application can be partitioned into multiple dependent tasks $\mathcal{N} = \{1, \dots, N\}$. In the edge computing system with the unknown system-side information, the UD makes offloading decisions to determine which edge servers should be selected for processing the tasks, aiming at minimizing the application delay under the long-term energy budget of the UD. Due to the heterogeneous nature of edge servers [55], the delay and energy consumption of the same task vary on different edge servers. Note that the first task (i.e., task 0) and the last task (i.e., task $N + 1$) are typically processed on the local UD to trigger the application and receive the computation results [9]. Such local processing is irrelevant to the offloading decision-making, and hence we do not consider the impact on task delay and energy consumption in this article. Main notations are illustrated in Table I.

A. Task Dependency Model

Task dependency refers to the logical precedence and data transfer among tasks, i.e., a task requires the outputs from the

TABLE I
MAIN NOTATIONS

Notation	Description
\mathcal{M}	Set of edge servers
\mathcal{N}	Set of dependent tasks
$T_{m,n}^{trans}$	The transmission delay of task n and edge server m
c_n	The computation workload of task n
f_m	The processing speed of edge server m
$T_{m,n}^{pro}$	The processing delay of task n
x_m^n	The offloading decision of task n
T_n^{start}	The start time of task n
$E(n)$	The transmission energy consumption of task n
$T(\alpha)$	The delay of rank α
$Q(\alpha)$	The energy deficit queue of rank α
K_m^α	The selected times of edge server m up to rank α
Ψ_m^n	The index function of edge server m
ξ_α^n	The selected edge server of task n in rank α
K_m^α	The selected times of edge server m up to rank α

preceding tasks, and its output needs to be fed into multiple subsequent tasks [9], [27]. For example, the ‘‘classification’’ tasks in a face recognition application rely on the outputs of the ‘‘feature extraction’’ tasks to complete classification. To illustrate such task dependency, we introduce a DAG [56], where \mathcal{N} denotes the task set, and (\mathcal{V}_i, n) represents the directed edge set. The directed edge set indicates that task $n \in \mathcal{N}$ can be processed after completing its preceding tasks $\forall i \in \mathcal{V}_i$; besides, when task n and task i are offloaded to different edge servers, the output of task i needs to be transferred to the edge server selected by task n . Without loss of generality, we assume that the tasks from the same application have the dependency, while the tasks from different applications are independent of each other [10].

B. Delay Model

1) *Transmission Delay*: It is comprised of transfer delay and access delay. On the one hand, the outputs of preceding tasks are transferred to the edge server m that is selected by the current task n through edge-to-edge connections (e.g., local-area network). The transfer incurs the transfer delay of $T_{m,n}^{trf}$, which relies on the hop distance along the shortest communication path [24]. Note that there is no transfer delay when the same edge server is selected by the current task and its preceding tasks. On the other hand, dependent task offloading produces access delay, which is determined by the task size and wireless transmission rate. When task n is offloaded from the UD to edge server m , the access delay is given by $T_{m,n}^{acc} = b_n/r_m$, where b_n indicates the task size (in bits) and r_m denotes transmission rate (in Mbps) of edge server m . The transmission rate is dependent on the signal to interference plus noise power ratio (SINR) between the UD and the selected edge server, $r_m = B_m \log_2(1 + SINR)$, where B_m represents the bandwidth of edge server m , SINR is primarily determined by various factors such as the transmission rate, channel bandwidth, channel gain, transmission power, and noise power between the UD and edge server m , respectively.

We assume that the processes of the data transfer and access are concurrent, then the transmission delay is denoted as $T_{m,n}^{trans} = \max\{T_{m,n}^{trf}, T_{m,n}^{acc}\}$.

2) *Processing Delay*: After a task has been transmitted to the selected edge server, the server will process the task with a specified processing speed. The processing delay is decided by task computation workload and edge processing speed. Let c_n denote the computation workload (in required CPU cycles) of task n , and f_m denotes the processing speed (in CPU frequency) of edge server m . Accordingly, the processing delay of task n is expressed as $T_{m,n}^{pro} = c_n/f_m$.

Combining the transmission delay and processing delay, we obtain the total delay of task n , which is given by:

$$T(n) = T_{m,n}^{trans} + T_{m,n}^{pro}. \quad (1)$$

Note that the system-side information, such as the transmission rate of r_m and the processing speed of f_m , is unknown to the UD in dependent task offloading [19]. Thus, the UD is incapable of obtaining the transmission delay and the processing delay. As a result, the task delay in (1) is unsolved. Without the delay, the UD has no prior knowledge of which edge server performs best (with the minimum task delay). Accordingly, an edge server with poor offloading performance (i.e., large task delay) may be selected. To address the issue, we propose to learn the edge offloading performance (e.g., task delay) to facilitate dependent task offloading in edge computing with the unknown system-side information (see details in Section IV).

C. Problem Formulation

Given a set of edge servers and dependent tasks, we denote a binary variable $x_m^n \in \{0, 1\}$ as the offloading decision of task n . When $x_m^n = 1$, task n is offloaded to edge server m ; otherwise, edge server m is not selected by task n . A feasible offloading solution needs to satisfy the following constraints.

1) *Task Dependency Constraint*: Recall the DAG defined in Section III-A, a task can be processed after its overall preceding tasks are completed. On this basis, we specify the start time of each task before making offloading decisions. Denote T_n^{start} as the start time of task n , requiring

$$T_n^{start} = \max_{i \in \mathcal{V}_i} \{T_i^{start} + T(i)\}, n \in \mathcal{N}. \quad (2)$$

2) *Long-Term Local Energy Budget*: When a UD offloads task n to the selected edge server, it consumes energy for transmission, denoted as $E(n)$. This energy consumption is determined by both the transmission power and access delay. Specifically, when task n is offloaded from the UD to edge server m , the transmission energy consumption can be expressed as $E(n) = P_{m,n} T_{m,n}^{acc}$, where $P_{m,n}$ denotes the transmission power required for offloading task n to edge server m . Constrained by the UD’s long-term energy budget of E , the transmission energy consumption needs to satisfy:

$$\sum_{n \in \mathcal{N}} E(n) \leq E. \quad (3)$$

3) *Limited Edge Computation Resources*: Suppose that task n consumes c_n edge computation resources for processing, the computation resources U_m of edge server m should satisfy:

$$\sum_{n \in \mathcal{N}_m} c_n \leq U_m, m \in \mathcal{M}. \quad (4)$$

where \mathcal{N}_m is the task set selecting edge server m concurrently.

4) *Offloading Decision Constraint*: To maintain task continuity [24], each task can only be offloaded to exactly one edge server for processing. Thus,

$$\sum_{m \in \mathcal{M}} x_m^n = 1, n \in \mathcal{N}. \quad (5)$$

Under the constraints of (2) through (5), the UD makes task offloading decisions to determine which edge servers should be selected for processing the current tasks, aiming at the minimum application delay. Let $\mathbf{x} = \{x_m^n, \forall n \in \mathcal{N}, m \in \mathcal{M}\}$ represent the offloading decisions of overall tasks, and denote the application delay as $T = \max_{n \in \mathcal{N}} \{T_n^{\text{start}} + T(n)\}$. Due to the unknown system-side information, the UD can only observe i.i.d. random application delay [19]. As such, we formulate the dependent task offloading problem as follows:

$$\begin{aligned} \mathbf{P1} : \quad & \min_{\mathbf{x}} \mathbb{E}\{T\}, \\ \text{s.t.} \quad & (2)-(5). \end{aligned} \quad (6)$$

It is non-trivial to solve **P1** directly. The reasons are that *i*) the offloading decision-making is highly coupled with task dependency. Such dependent task offloading has been proved to be NP-hard even in simplified scenarios [56]; *ii*) the long-term energy constraint of the UD restricts the short-term task offloading decisions. *iii*) the unknown system-side information demands learning-based offloading, while task dependency hinders continuous learning and thus incapacitates the existing learning-based algorithms. To tame these challenges, we design an efficient online learning-based algorithm by jointly considering task dependency and unknown system-side information under the long-term energy budget constraint of the UD. In the following section, we will present our algorithm in detail.

IV. ONLINE LEARNING-BASED ALGORITHM FOR DEPENDENT TASK OFFLOADING

In this section, we first adopt a BFS method to decouple task dependency, then we transfer the long-term offloading problem to an online optimization problem via leveraging the Lyapunov optimization technique. On this basis, we propose the OL-DTO algorithm based on the MAB theory. The algorithm enables to address the unknown system-side information and is augmented by task dependency awareness. At last, we derive the performance results for the proposed algorithm.

A. Decouple Task Dependency

We decouple task dependency based on the DAG. Taking a real-world Gaussian elimination application [56] as an instance, its DAG with four matrixes is depicted in Fig. 2(a). Each circle represents a task, and the directed edges reflect the logical precedence and the data transfer among tasks (i.e., dependency). In the DAG, tasks 1 to 7 need to be traversed exactly once under the task dependency constraints. Note that task 0 and task 8 are the first and last tasks that are processed on local UD and we do not account for them. These features enable the accommodation of breadth-first-search (BFS) algorithm over

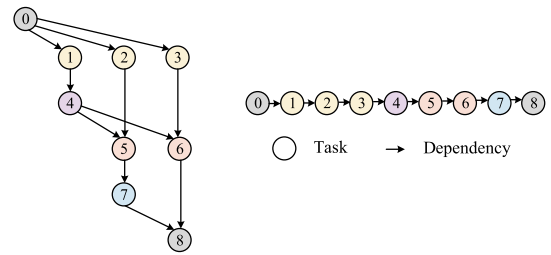


Fig. 2. Task dependency.

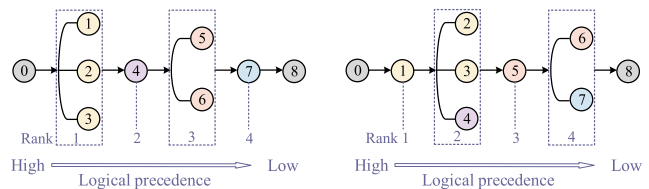


Fig. 3. Concurrent-enhanced task offloading.

the DAG [54], so that we can retrieve the task offloading order by recording the search path. In this way, task dependency is decoupled as a sequential relationship as shown in Fig. 2(b). Such a relationship is straightforward as one task is assumed to have only one preceding task and one subsequent task. In the sequential task offloading, we obtain the GE application's delay, i.e., the total delay of all tasks.

To further reduce the application delay, we optimize the BFS method and propose a concurrent-enhanced dependency decouple algorithm, as presented in Algorithm 1. In the algorithm, we determine *start ranks* and *end ranks* for tasks in the application, so that we can ascertain at which ranks the tasks can be processed and need to be completed. To achieve this, we first recognize the tasks with zero preceding tasks currently as the same start rank for the guarantee of logical precedence across ranks. The results are presented in Fig. 3(a). Yet, we observe from Fig. 3(a) that task 4 can be processed after tasks 1, 2, and 3 are all completed, while task 4 is independent of tasks 2 and 3 as shown in Fig. 2(a). Thus, we then need to determine the tasks' end ranks to further enhance concurrent task offloading. To this end, we adjust the rank order for the tasks that have no dependency on the tasks in their subsequent ranks. Such adjustment does not change the number of ranks. The corresponding results are presented in Fig. 3(b). Note that the results indicate the maximum end ranks as the tasks may be completed before the ranks. Combining Fig. 3(a) and (b), we summarize the start rank and the maximum end rank of tasks in the GE application with four matrixes in Table II.

Aiming to ascertaining the actual end ranks for tasks, we introduce *rank delay* as follows. Note that end ranks refers to the actual end ranks in our work. For a specific rank $\alpha \in \mathcal{A}$, its delay is determined by two aspects. One aspect is the delay performed by the tasks of which the start ranks and end ranks

TABLE II
START AND END RANKS OF TASKS IN THE GE APPLICATION

Rank \ Task	$n = 1$	2	3	4	5	6	7
$\alpha = 1$	-	-	-				
$\alpha = 2$		-	-	-			
$\alpha = 3$			-		-	-	
$\alpha = 4$						-	-

are both the same as rank α . Thus, the tasks can be completed in the single rank, and we denote the delay as T_α^{one} . The other aspect is the delay conducted by the tasks of which the start ranks are different with rank α while the end ranks are the same as rank α . In this case, the tasks are processed across multiple ranks, and we denote the delay as T_α^{mul} . Furthermore, we define the remaining ranks (not the start and end ranks) that tasks go through as the *middle ranks* of the tasks. For example, rank 2 is the middle rank of task 3 in Table II. If the delay of a task is smaller than all the ranks' delay between the start rank and a middle rank, the middle rank will be replaced by the task's end rank. Guided by this, we enable the acquisition of start ranks and (actual) end ranks for all tasks. Then, we denote the tasks of which the start ranks, middle ranks and end ranks are rank α as \mathcal{N}_α^{sta} , \mathcal{N}_α^{mid} , and \mathcal{N}_α^{end} , respectively. Let α_n^s represent the start rank of task n . Thus, we have

$$T_\alpha^{one} = \max\{T(n), \forall n \in \mathcal{N}_\alpha^{sta} \& n \in \mathcal{N}_\alpha^{end}\}, \quad (7)$$

$$T_\alpha^{mul} = \max\{T(n) - \sum_{i=\alpha_n^s}^{\alpha-1} T(i), \forall n \notin \mathcal{N}_\alpha^{sta} \& n \in \mathcal{N}_\alpha^{end}\}. \quad (8)$$

Accordingly, we obtain the delay of rank α as

$$T(\alpha) = \max\{T_\alpha^{one}, T_\alpha^{mul}\}. \quad (9)$$

Tasks make offloading decisions in the start ranks, denoting x_m^α as the offloading decision set of tasks which start ranks are rank α . Guided by Algorithm 1, we obtain the application delay that is the total delay of overall ranks, i.e., $T = \sum_{\alpha=1}^A \mathbb{E}\{T(\alpha)\}$. We decouple **P1** as follows:

$$\begin{aligned} \mathbf{P2}: \quad & \min_{x_m^\alpha} \frac{1}{A} \sum_{\alpha=1}^A \mathbb{E}\{T(\alpha)\}, \quad (10) \\ \text{s.t.} \quad & (2)-(5). \end{aligned}$$

A major challenge for solving **P2** is that the long-term energy constraint of the UD shown in (3) hinders the derivation of offloading decisions.

Computational complexity. Finding out the task with zero preceding task in line 5 incurs $\mathcal{O}(N)$ computational complexity. After adding these tasks to the start rank queue, we need to update the dependency by deleting the directed edges of the tasks in the start rank queue. The corresponding computational complexity of the update is $\mathcal{O}(V_e)$, where V_e is the number of directed edges in the DAG. Additionally, finding out the tasks that are independent with the overall tasks in the subsequent start ranks in line 11 incurs $\mathcal{O}(N^2)$ computational complexity. Then, we need to adjust and update the end rank for the

Algorithm 1: Concurrent-Enhanced Dependency Decouple Algorithm.

Input: DAG.

Output: Rank queue.

1: **Phase 1: Initialization**

2: Set a start rank queue as zero.

3: Set an end rank queue as zero.

4: Determine the dependency and the number of preceding tasks for each task based on the DAG.

5: **Phase 2: Start Rank Determination**

6: Find out the tasks with zero preceding task.

7: Add the tasks to the start rank queue.

8: Update the task dependency and number of preceding tasks.

9: Back to line 5 for iteration until each task can be found in the start rank queue.

10: **Phase 3: End Rank Determination**

11: Find out the tasks that are independent with the overall tasks in the subsequent start ranks.

12: Update the end ranks of the tasks by moving the tasks from their start ranks to the subsequent ranks.

13: Back to line 11 for iteration until the tasks have dependency with at least one tasks in the subsequent end ranks.

14: Compare the task delay and rank delay between the start rank and the middle rank.

15: Replace the middle ranks as the end ranks when the task delay is smaller than the rank delay.

tasks, which incurs $\mathcal{O}(N)$ computational complexity. At last, the comparison between task delay and rank delay incurs $\mathcal{O}(NA)$ computational complexity, where $A = |\mathcal{A}|$ is the number of ranks. Therefore, the computational complexity of Algorithm 1 is $\mathcal{O}(V_e + N^2 + NA)$.

B. Online Problem Transformation

To address the long-term energy constraint, we leverage the Lyapunov optimization technique [57] to transfer **P2** to an online optimization problem. To achieve this, we construct a virtual energy deficit queue, representing a historical measurement of the exceeded UD's energy consumption. The energy deficit queue of rank α is defined as:

$$Q(\alpha) = \max\{Q(\alpha - 1) + E(\alpha) - \bar{E}(\alpha), 0\}, \quad (11)$$

where $E(\alpha) = \sum_{n \in \mathcal{N}_\alpha^{sta}} E(n)$ represents the energy consumption of rank α as tasks are transmitted to edge servers in the start ranks, $\bar{E}(\alpha) = EN_\alpha^{sta}/N$ is the UD's energy budget of rank α , where $N_\alpha^{sta} = |\mathcal{N}_\alpha^{sta}|$ is the number of tasks of which start ranks are rank α . Additionally, the initial deficit queue $Q(0)$ is set as zero.

Combining the energy deficit queue with **P2**, we utilize the *Lyapunov drift-plus-penalty framework* to formulate the following online dependent task offloading problem:

$$\mathbf{P3}: \quad \min_{x_m^\alpha} VT(\alpha) + E(\alpha)Q(\alpha), \quad (12)$$

Algorithm 2: Online Transformation Algorithm.

Input : DAG, V , b_n and c_n .
Output: Offloading decisions.

- 1 **Phase 1: Initialization**
- 2 Default $T(0) = 0$, $E(0) = 0$ and $Q(0) = 0$.
- 3 **Phase 2: Offloading Decision Making**
- 4 Decouple task dependency via Algorithm 1.
- 5 **for** $n = 1$ **to** $n = N$ **do**
- 6 Require delay $T(\alpha)$ and energy consumption $E(\alpha)$ under different offloading decisions.
- 7 Find out the offloading decisions by solving the online dependent task offloading problem **P3**.
- 8 Update the energy deficit queue based on (11).
- 9 **end**

$$\text{s.t.} \quad (2), (4), (5),$$

where V is a positive control parameter to balance the application delay and local energy consumption, and $Q(\alpha)$ guides the real-time local energy consumption. A larger deficit queue $Q(\alpha)$ indicates less remaining local energy, the UD then intends to reduce its energy consumption. By this means, an online energy adjustment can be realized in **P3**. For ease of exposition, we define $\Phi(\alpha) \triangleq VT(\alpha) + E(\alpha)Q(\alpha)$.

An online problem transformation algorithm is presented in Algorithm 2, which transfers the long-term offloading problem **P2** to an online optimization problem **P3**. However, we emphasize that even with the proposed Algorithm 2, solving **P3** is still not straightforward as the system-side information is unknown, making it difficult to acquire $T(\alpha)$ and energy consumption $E(\alpha)$ for different offloading decisions. Therefore, we need to explore learning-based solutions to develop an online algorithm that can efficiently solve this problem. In the next subsection, we will discuss this approach in more detail.

C. Learning-Based Solutions

In this subsection, we propose a learning-based solution to address the unknown system-side information. To achieve this, we formulate **P3** as an MAB problem [58], where a UD acts as the ‘‘gambler,’’ and edge servers perform as ‘‘arms.’’ The UD makes offloading decisions based on the historical observations of edge offloading performance. In such offloading, the UD selects either a new edge server (i.e., *exploration*) or the optimal edge server up to now (i.e., *exploitation*) for processing subsequent tasks. It is crucial for learning-based algorithms to achieve the balance between exploration and exploitation. The reasons are that excessive exploration slows down the convergence and causes degraded offloading performance, while exaggerated exploitation inevitably abandons potential better edge servers and is easily trapped by poor offloading decisions.

Although the existing MAB algorithms (e.g., UCB [58]) achieve well-balanced exploration and exploitation through continuous learning, the algorithms cannot be directly applied to dependent task offloading. Recall Section IV-A, where dependent tasks are categorized into different ranks. On the one

hand, tasks in the same start rank are offloaded concurrently and cannot learn the edge offloading performance from each other’s offloading. This hence hampers continuous learning for tasks in learning-based algorithms. On the other hand, if we implement a learning-based algorithm for each start rank, continuous learning can be satisfied but the learning efficiency will be greatly impaired. This is because the start rank cannot reflect the end ranks of tasks, impeding the update of edge offloading performance in time. Despite taking end ranks into account, the UD has to randomly select a task in the end rank for the update, while the update neglects the effects of most tasks’ offloading in the same end rank on the learning efficiency. Furthermore, when tasks are offloaded to different edge servers, the preceding tasks’ outputs will be transferred between the neighboring ranks. Large transfer data adversely impacts the application delay. In this case, the learning-based algorithm demands exploitation rather than exploration to speed up the algorithm’s convergence. Since the existing MAB algorithms fail to properly tackle the above-mentioned issues, the exploration-exploitation problem will be exacerbated in dependent task offloading.

Motivated by the limitations of the existing algorithms, we propose the OL-DTO algorithm, as presented in Algorithm 3. The algorithm enables to address the unknown system-side information and is augmented by task dependency awareness. Specifically, tasks are offloaded to the edge servers based on the historical edge offloading performance observations of start rank α , the edge offloading performance in rank α is then updated according to the offloading decisions of overall tasks of which the end rank is rank α . In this way, the OL-DTO algorithm maintains continuous learning for each start rank while emphasizing the learning efficiency of tasks in the same end rank. Moreover, we introduce a dependency factor into the algorithm to avoid unnecessary exploration. Overall, the OL-DTO algorithm is extremely different from the existing algorithms and is capable of dependent task offloading.

The OL-DTO algorithm consists of two phases, i.e., the *initialization* phase (lines 1–3) and the *learning while offloading* phase (lines 4–22). In the initialization phase, we set the initial task delay, energy consumption, and energy deficit queue as zero. Besides, we denote K_m^α as the selected times of edge server m up to rank α and define $\bar{\Phi}_m^\alpha = \sum_{i=1}^{i=\alpha} \Phi(\alpha) / K_m^\alpha$. Both K_m^α and $\bar{\Phi}_m^\alpha$ are defaulted as zero when ‘*learning while offloading*’ is not implemented. In the ‘*learning while offloading*’ phase, we first decouple task dependency through Algorithm 1. Then, we transfer the long-term offloading problem to the online optimization problem via Algorithm 2. Following that, the UD makes offloading decisions for dependent tasks based on the previously observed edge offloading performance. Let ξ_α^n represent the selected edge server for processing task n in start rank α . Each edge server will be selected at least once to observe its offloading performance (i.e., lines 7–10). Note that only noisy edge offloading performance can be observed due to the complicated wireless environment and varying computation workloads [19], [22]. For this reason, we introduce an index function to estimate the edge offloading performance after each edge server has been selected once. Given task n in start rank α , the index function

Algorithm 3: OL-DTO Algorithm.

Input : DAG, V , b_n , c_n , $\tilde{\delta}_{i,j}$, δ_m and β .
Output: Offloading decision ξ_α^n .

- 1 **Phase 1: Initialization**
- 2 Default $T(0) = 0$, $E(0) = 0$ and $Q(0) = 0$.
- 3 Set $K_m^0 = 0$ and $\bar{\Phi}_m^0 = 0$.
- 4 **Phase 2: Learning while offloading**
- 5 Decouple task dependency via Algorithm 1.
- 6 Transfer the long-term offloading **P2** to an online optimization problem **P3** using Algorithm 2.
- 7 **for** $n = 1$ **to** $n = N$ **do**
- 8 **if** \exists edge server m , $K_m^{\alpha-1} = 0$ **then**
- 9 Select edge server m for processing task n
 in start rank α , i.e., $\xi_\alpha^n = m$ and $K_{\xi_\alpha^n}^n = 1$.
- 10 **end**
- 11 **else**
- 12 **for** rank $\alpha = 1$ **to** rank $\alpha = A$ **do**
- 13 **for** tasks belongs to start rank α **do**
- 14 Calculate the index value for each
 edge server based on (13).
- 15 Select edge server ξ_α^n for task n in
 start rank α to satisfy (14).
- 16 Retain $\Phi(\alpha)$ for the rank α .
- 17 **end**
- 18 Update $\bar{\Phi}_m^\alpha$ based on (15).
- 19 Update K_m^α based on (16).
- 20 **end**
- 21 **end**
- 22 **end**

of edge server m is presented as:

$$\Psi_m^n = \bar{\Phi}_m^{\alpha-1} - \sqrt{\frac{\beta(1 - \tilde{\delta}_n) \ln \alpha}{(1 - \delta_m) K_m^{\alpha-1}}}, \quad (13)$$

where parameter β is used to adjust the weight of exploration. The dependency factor of $\tilde{\delta}_n \in [0, 1]$ is the normalized transferred data of task n . A larger dependency factor facilitates exploitation to avoid large transfer delay. Additionally, we denote $\delta_m \in (0, 1)$ as the observation variance of edge server m , which can be reasonably inferred based on historical measurements and experimental results [26], [59]. A larger variance indicates larger offloading uncertainties and thus needs more exploration. We assume that the variance is no less than the dependency factor as the variance affects both the application delay and the local energy consumption.

Accordingly, task n in start rank α is offloaded to edge server ξ_α^n based on the index-based minimum value research:

$$\xi_\alpha^n = \arg \min_{m \in M} \Psi_m^n. \quad (14)$$

When overall tasks in start rank α are offloaded to the selected edge servers, we update $\bar{\Phi}_m^\alpha$ based on the tasks of which the end ranks are α :

$$\bar{\Phi}_m^\alpha = \sum_{n \in \mathcal{N}_\alpha^{\text{end}}} \frac{\bar{\Phi}_m^{\alpha-1} K_m^{\alpha-1} + \Phi_m^n}{K_m^{\alpha-1} + \mathbb{I}\{\xi_\alpha^n = m\}}, m = 1, \dots, M, \quad (15)$$

where $\mathbb{I}\{x\}$ is an indicator function. When event x is true, $\mathbb{I}\{x\} = 1$; otherwise, $\mathbb{I}\{x\} = 0$. Recall the task offloading decision of x_m^n in Section III-C, $\Phi_m^n = VT(x_m^n) + E(x_m^n)Q(x_m^n)$. Correspondingly, we update the selected times of edge server m up to rank α as follows:

$$K_m^\alpha = K_m^{\alpha-1} + \sum_{n \in \mathcal{N}_\alpha^{\text{end}}} \mathbb{I}\{\xi_\alpha^n = m\}, m = 1, \dots, M. \quad (16)$$

Note that the OL-DTO algorithm is augmented by task dependency awareness. Specifically, guided by the tasks' start ranks and end ranks, it maintains continuous learning for each start rank while emphasizing the learning efficiency of tasks in the same end rank; besides, the dependency factor speeds up the algorithm's convergence. Thus, the algorithm achieves the balance between exploration and exploitation in learning-based dependent task offloading.

Computational complexity. Line 14 calculates the index value for edge servers, the computational complexity is $\mathcal{O}(M)$. Line 15 shows a minimum value-seeking problem, occupying $\mathcal{O}(M)$. The update behaviors, such as line 18 and 19, have a computational complexity $\mathcal{O}(1)$. Therefore, we conclude that the computational complexity of Algorithm 3 is $\mathcal{O}(M)$ for processing a single task. Based on this, we obtain the total computational complexity is $\mathcal{O}(NM)$.

D. Performance Analysis

In this subsection, we analyze the performance of OL-DTO algorithm on application delay and local energy consumption.

We first derive the following theorem to illustrate the performance gap of Algorithm 1.

Theorem 1. Let x_{dd}^α be the concurrent-enhanced dependency-decoupled offloading decisions of tasks in rank α , which is performed by Algorithm 1. We obtain:

$$\frac{1}{A} \sum_{\alpha=1}^A \mathbb{E}\{T(x_{dd}^\alpha)\} \leq T_{p_1}^{\text{opt}} + A\epsilon, \quad (17)$$

where $T_{p_1}^{\text{opt}}$ denotes the optimal task delay of **P1**, and ϵ is the maximum delay gap among tasks in a single end rank.

Proof. Algorithm 1 decouples task dependency by determining the tasks' start ranks and end ranks. Specifically, the tasks are transmitted to edge servers in the start rank and need to be completed in the end rank. Guided by this, the application delay can be presented by the sum of end ranks' delay. For a specific end rank, the delay is the maximum delay of tasks in the end rank. Since the tasks in the end rank may not be completed simultaneously, we denote ϵ as the maximum delay gap among the tasks for a single end rank. Thus, we acquire the delay gap for A ranks between **P1** and **P2** is $A\epsilon$.

Furthermore, the tasks are transmitted to edge servers in start ranks and generate local transmission energy consumption. For a specific start rank, the energy consumption is the sum energy consumption of tasks in the start rank, which is irrelevant to the rank partition. Therefore, there is no energy consumption gap between **P1** and **P2**.

We then analyze learning regret. Denote the learning regret of edge server m as $\theta_m = \nu_m - \nu_*$, where $\nu_m = \mathbb{E}\{\Phi_m^n\}$ and

$\nu_* = \min\{\nu_m, \forall m \in \mathcal{M}\}$. As such, the total learning regret for A ranks is expressed as:

$$\Theta = \sum_{m=1}^M K_m^A \theta_m, \quad (18)$$

of which the upper bound is presented in the following theorems.

Theorem 2. The upper bound of the learning regret performed by the OL-DTO algorithm is:

$$\Theta \leq \sum_{m=1}^M \left(\frac{8 \ln A}{\theta_m} + \theta_m + \frac{\theta_m}{3} \pi^2 \right). \quad (19)$$

Proof. Based on the analysis in [58], we derive the learning regret of the OL-DTO Algorithm by bounding K_m^A . To this end, we define that each edge server has been selected once after M_r ranks. Suppose τ is a positive integer, we obtain

$$\begin{aligned} K_m^A &= 1 + \sum_{\alpha=M_r}^A x_m^\alpha \leq \tau + \sum_{\alpha=M_r}^A \{x_m^\alpha, K_m^{\alpha-1} \geq \tau\} \\ &\leq \tau + \sum_{\alpha=M_r}^A \left\{ \bar{\Phi}_*^{K_m^{\alpha-1}} - \sqrt{\frac{\beta(1-\tilde{\delta}_n) \ln(\alpha-1)}{(1-\delta_m) K_m^{\alpha-1}}} \right\} \\ &\geq \bar{\Phi}_m^{K_m^{\alpha-1}} - \sqrt{\frac{\beta(1-\tilde{\delta}_n) \ln(\alpha-1)}{(1-\delta_m) K_m^{\alpha-1}}, K_m^{\alpha-1} \geq \tau} \\ &\leq \tau + \sum_{\alpha=M_r}^A \left\{ \max_{0 < s < \alpha} \bar{\Phi}_*^s - \sqrt{\frac{\beta(1-\tilde{\delta}_n) \ln(\alpha-1)}{(1-\delta_m)s}} \right\} \\ &\geq \min_{\tau \leq s_m \leq \alpha} \left\{ \bar{\Phi}_m^{s_m} - \sqrt{\frac{\beta(1-\tilde{\delta}_n) \ln(\alpha-1)}{(1-\delta_m)s_m}} \right\} \\ &\leq \tau + \sum_{\alpha=1}^{\infty} \sum_{s=1}^{\alpha-1} \sum_{s_m=\tau}^{\alpha-1} \left\{ \bar{\Phi}_*^s - \sqrt{\frac{\beta(1-\tilde{\delta}_n) \ln \alpha}{(1-\delta_m)s}} \right\} \\ &\geq \bar{\Phi}_m^{s_m} - \sqrt{\frac{\beta(1-\tilde{\delta}_n) \ln \alpha}{(1-\delta_m)s_m}} \}. \end{aligned} \quad (20)$$

When $\bar{\Phi}_*^s - \sqrt{\frac{\beta(1-\tilde{\delta}_n) \ln \alpha}{(1-\delta_m)s}} \geq \bar{\Phi}_m^{s_m} - \sqrt{\frac{\beta(1-\tilde{\delta}_n) \ln \alpha}{(1-\delta_m)s_m}}$ is satisfied, at least one of the following inequalities must hold [58]:

$$\bar{\Phi}_*^s \geq \nu_* + \sqrt{\frac{\beta(1-\tilde{\delta}_n) \ln \alpha}{(1-\delta_m)s}}, \quad (21)$$

$$\bar{\Phi}_m^{s_m} \leq \nu_m - \sqrt{\frac{\beta(1-\tilde{\delta}_n) \ln \alpha}{(1-\delta_m)s_m}}, \quad (22)$$

$$\nu_* > \nu_m - 2\sqrt{\frac{\beta(1-\tilde{\delta}_n) \ln \alpha}{(1-\delta_m)s_m}}. \quad (23)$$

On this basis, we investigate the rationality of In (21) through (23). From (23), we derive that $s_m < \frac{4\beta(1-\tilde{\delta}_n) \ln \alpha}{(1-\delta_m)(\theta_m)^2}$. Additionally, we noticed that $\tau \leq s_m < \alpha$ based on the definition of (20). If we arbitrarily give $\forall \tau = \lceil \frac{4\beta(1-\tilde{\delta}_n) \ln \alpha}{(1-\delta_m)(\theta_m)^2} \rceil$, then $s_m \geq \frac{4\beta(1-\tilde{\delta}_n) \ln \alpha}{(1-\delta_m)(\theta_m)^2}$.

Clearly, the conclusion contradicts the former one from (23). As such, we recognize that the (23) is illegal.

Next, we analysis the expressions (21) and (22) based on Chernoff-Hoeffding bound.

$$\mathbb{P} \left\{ \bar{\Phi}_*^s \geq \nu_* + \sqrt{\frac{\beta(1-\tilde{\delta}_n) \ln \alpha}{(1-\delta_m)s}} \right\} \leq \alpha^{-\beta^2 \left(\frac{1-\tilde{\delta}_n}{1-\delta_m} \right)^2}. \quad (24)$$

$$\mathbb{P} \left\{ \bar{\Phi}_m^{s_m} \leq \nu_m - \sqrt{\frac{\beta(1-\tilde{\delta}_n) \ln \alpha}{(1-\delta_m)s_m}} \right\} \leq \alpha^{-\beta^2 \left(\frac{1-\tilde{\delta}_n}{1-\delta_m} \right)^2}. \quad (25)$$

Given $\beta = 2$, based on the above analysis, we obtain:

$$\begin{aligned} \mathbb{E}(K_m^A) &\leq \lceil \frac{8(1-\tilde{\delta}_n) \ln A}{(1-\delta_m)(\theta_m)^2} \rceil + \sum_{\alpha=1}^{\infty} \sum_{s=1}^{\alpha-1} \sum_{s_m=\tau}^{\alpha-1} 2\alpha^{-4 \left(\frac{1-\tilde{\delta}_n}{1-\delta_m} \right)^2} \\ &\leq \frac{8(1-\tilde{\delta}_n) \ln A}{(1-\delta_m)(\theta_m)^2} + 1 + \sum_{\alpha=1}^{\infty} \sum_{s=1}^{\alpha} \sum_{s_m=1}^{\alpha} 2\alpha^{-4} \\ &\leq \frac{8 \ln A}{(\theta_m)^2} + 1 + \frac{\pi^2}{3}. \end{aligned} \quad (26)$$

Above all, we obtain the upper bound of the learning regret, which is denoted as follows:

$$\Theta \leq \sum_{m=1}^M \left(\frac{8 \ln A}{\theta_m} + \theta_m + \frac{\theta_m}{3} \pi^2 \right). \quad (27)$$

Based on the bound, we further analyze the performance of the OL-DTO algorithm on the application delay and local energy consumption under the Lyapunov framework.

Theorem 3. Let x_{ol}^α be online learning-based offloading decisions of tasks in rank α , which is performed by Algorithm 3. Given a positive control parameter V , we obtain:

$$\frac{1}{A} \sum_{\alpha=1}^A \mathbb{E}\{T(x_{ol}^\alpha)\} \leq T^{opt} + \frac{1}{V} \left(Q + \frac{\Theta}{A} + \epsilon \right), \quad (28)$$

$$\frac{1}{A} \sum_{\alpha=1}^A \mathbb{E}\{Q(x_{ol}^\alpha)\} \leq \frac{1}{\eta} \left(V(T_{sys}^{max} - T^{opt}) + Q + \frac{\Theta}{A} \right), \quad (29)$$

where $T^{opt} = \frac{1}{A} \sum_{\alpha=1}^A \mathbb{E}\{T(x_*^\alpha)\}$ denotes the delay under the optimal solution of x_*^α to **P2**. However, the optimal solution is hard to obtain in practical edge computing since it relies on the deterministic system-side information and long-term UD's energy consumption. Additionally, $Q = \frac{1}{2}(E^{max} - \bar{E}(\alpha))^2$, E^{max} and T_{sys}^{max} are the maximum local energy consumption and delay that tasks in a rank can tolerate, respectively.

Proof. We introduce a quadratic Lyapunov function to measure the "size" of an energy deficit queue [57], which is defined as $L(Q(\alpha)) \triangleq \frac{1}{2}Q^2(\alpha)$. A larger function reflects less remaining UD's energy for subsequent ranks. Then, we introduce a one-slot

conditional Lyapunov drift as

$$\begin{aligned} \Delta(Q(\alpha)) &\triangleq \mathbb{E}\{L(Q(\alpha+1)) - L(Q(\alpha))\} \\ &\leq \mathbb{E}\left\{\frac{1}{2}Q^2(\alpha+1) - \frac{1}{2}Q^2(\alpha)\right\} \\ &\leq \mathbb{E}\{Q(\alpha)(E(\alpha) - \bar{E}(\alpha))\} + Q, \end{aligned} \quad (30)$$

Based on Theorem 4.5 in [57], we present the two lemmas:

Lemma 1. Give an arbitrary positive parameter λ , there is a stationary and randomized offloading decision set x_{Π}^{α} for **P2**,

$$\mathbb{E}\{T(x_{\Pi}^{\alpha})\} \leq T^{opt} + \lambda, \quad (31)$$

$$\mathbb{E}\{E(x_{\Pi}^{\alpha}) - \bar{E}(x_{\Pi}^{\alpha})\} \leq \lambda. \quad (32)$$

Lemma 2. Exist a positive parameter η , there is a offloading decision set x_{γ}^{α} for **P2** that satisfies the following expressions:

$$\mathbb{E}\{T(x_{\gamma}^{\alpha})\} = \chi(\eta), \quad (33)$$

$$\mathbb{E}\{E(x_{\gamma}^{\alpha}) - \bar{E}(x_{\gamma}^{\alpha})\} \leq -\eta. \quad (34)$$

Based on Lemma 1, we derive the following inequalities:

$$\Delta(Q(x_{ol}^{\alpha})) + V\mathbb{E}\{T(x_{ol}^{\alpha})\} \leq V(T^{opt} + \lambda) + Q + \Theta_{\alpha} + \epsilon, \quad (35)$$

where Θ_{α} is the learning regret of rank α . Then, we obtain the optimality gap on the application delay by summing (35) overall ranks, dividing VA on each side, and letting λ go to zero. We have

$$\frac{1}{A} \sum_{\alpha=1}^A \mathbb{E}\{T(x_{ol}^{\alpha})\} \leq T^{opt} + \frac{1}{V} \left(Q + \frac{\Theta}{A} + \epsilon \right). \quad (36)$$

Following this, we bound the long-term energy deficit queue of the UD. Based on Lemma 2, we derive:

$$\Delta(Q(x_{ol}^{\alpha})) + V\mathbb{E}\{T(x_{ol}^{\alpha})\} \leq V\chi(\eta) - \eta Q(x_{ol}^{\alpha}) + Q + \Theta_{\alpha}. \quad (37)$$

We sum the above (37) over $\alpha \in \{1, \dots, A\}$ and divides $A\eta$ on each side. We obtain

$$\frac{1}{A} \sum_{\alpha=1}^A \mathbb{E}\{Q(x_{ol}^{\alpha})\} \leq \frac{1}{\eta} \left(V(T_{sys}^{max} - T^{opt}) + Q + \frac{\Theta}{A} \right). \quad (38)$$

From Theorem 3, we find that an $[O(1/V), O(V)]$ trade-off exists between the application delay and the local energy consumption. When V tends to ∞ , the minimum application delay can be achieved at the price of a large UD's energy violation. By rationally adjusting the control parameter V , the OL-DTO algorithm can realize the well-balanced application delay and local energy consumption.

V. PERFORMANCE EVALUATION

The simulation experiments are run on Dev-C++ 5.11 and Matlab R2014a hosted by a PC with an Intel i7 2.5 GHz CPU, and 16 GB RAM. We evaluate the performance of our proposed algorithm using real-world applications (from [56]) and real measurements (from [60]) under varying task number, DAG structures, online control parameters, and online learning factors.

TABLE III
PARAMETERS SETTING

Key Parameter	Value
Number of edge servers, M	10
Task size, b_n	[500, 1500] Kb
Required CPU cycles, c_n	[0.5, 1.5] GHz
Edge processing speed, f_m	[4, 8] GHz
Data transmission rate, r_m	5 GHz
Transmission power, $P_{m,n}$	100 mW
Computation resource consumption, u_n	[0.5, 1]
Edge computation resources, U_m	[2, 5]
Transfer delay, $T_{m,n}^{trf}$	(50, 100) ms
Observation variance, δ_m	[0.2, 0.3]
Dependency factor, \tilde{o}_n	[0.1, 0.2]

A. Evaluation Setup

We consider an MEC network consisting of 10 edge servers, with several UDs generating dependent tasks, and the number of tasks n varies between 30 and 220 [6]. We use the GE application and Fast Fourier Transform (FFT) application to generate the real-world DAG structures [56]. Based on real measurements in [60], the task size of b_n is distributed in [500, 1500] Kb, and the required CPU cycles of c_n are drawn from [0.5, 1.5] GHz; the edge processing speed of f_m is distributed in [4, 8] GHz, and the data transmission rate of r_m is drawn from [9, 11] Mbps in 4G networks. Note that f_m and r_m are not prior knowledge for UDs in edge computing with unknown system-side information. A UD offloads its tasks to edge servers with 100 mW transmission power. Each task consumes $u_n \in [0.5, 1]$ computation resources for processing, and edge server m has $U_m \in [2, 5]$ computation resources [19]. The transfer delay is in (50, 100) ms. Additionally, we assume that the observation variance of an edge server is distributed in [0.2, 0.3], and the dependency factor is set in [0.1, 0.2]. The key parameters used in the simulations are listed in Table III.

We further compare the performance of our proposed OL-DTO algorithm with the following baselines:

- *Local algorithm:* Overall tasks are processed on the resource-limited UD sequentially. The parameters (e.g., local processing capabilities) are set based on [9].
- *Sequential algorithm:* It is proposed in [26], where tasks are assumed to have a sequential relationship and are offloaded successively in the learning-based algorithm.
- *Energy-myopic algorithm:* A hard single-rank energy constraint is imposed for dependent task offloading, rather than following the long-term energy budget constraint of the UD. The Energy-myopic algorithm is widely used in literature such as [61], [62] to perform as a baseline.
- *UCB-D algorithm:* It randomly chooses a task and updates the rank's edge offloading performance based on the selected task's offloading. The UCB-D algorithm formulates the offloading decision-making problem as the index-based minimum value research, and the index function is defined as $\Phi_m^{\alpha-1} - \sqrt{\beta \ln \alpha / K_m^{\alpha-1}}$ [58].
- *Optimal policy:* It is performed by an omniscient oracle with the deterministic system-side information and long-term energy consumption [59]. The policy serves as an ideal performance benchmark for our proposed algorithm.

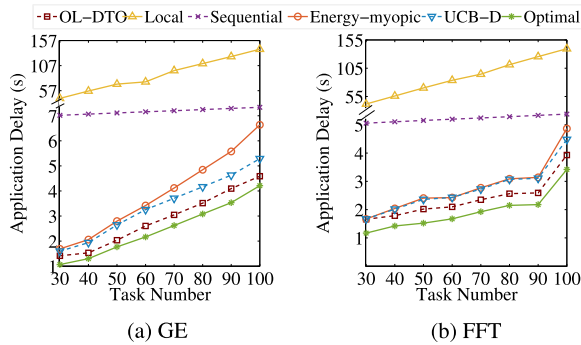


Fig. 4. Application delay under the (a) GE and (b) FFT.

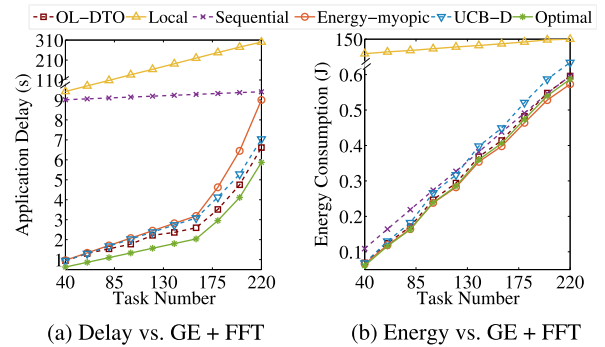


Fig. 6. Application delay and energy consumption.

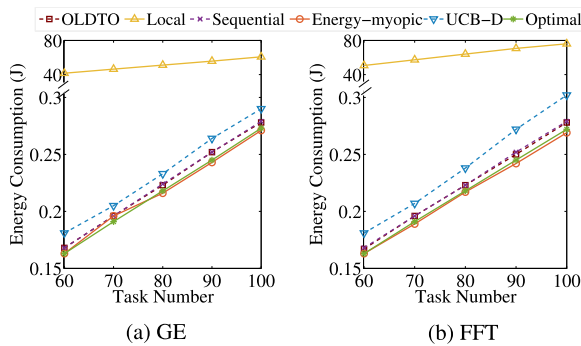


Fig. 5. Local energy consumption under the (a) GE and (b) FFT.

B. Performance of Dependent Task Offloading

Fig. 4 depicts the application delay under the two DAG structures, i.e., GE and FFT. Compared with GE, FFT has fewer ranks for the same number of tasks (see [56] for more details). Thus, more tasks in FFT can be processed concurrently, and the application delay of FFT is smaller than that of GE. For example, when there are 100 tasks, the application delay of GE is 4.588, 6.639, 5.289, and 4.212 seconds performed by the OL-DTO, Energy-myopic, UCB-D, and Optimal algorithms, respectively; while the application delay of FFT is 3.931, 4.866, 4.481, 3.421 seconds under the OL-DTO, Energy-myopic, UCB-D, and Optimal algorithms, respectively. Besides, for 100 tasks, it can be found that the proposed OL-DTO algorithm decreases the application delay in GE by 96.71%, 80.76%, 30.89%, and 13.26% compared with the Local, Sequential, Energy-myopic, and UCB-D algorithms, respectively. Fig. 5 depicts the local energy consumption under the DAG structures of GE and FFT. Particularly, when there are 100 tasks for the UCB-D algorithm, its energy consumption in FFT is increased by 4.14% compared to that in GE; while 0.74% under the Energy-myopic algorithm. The reason is that the UCB-D algorithm has no dependency awareness and demands more tasks to explore better edge servers, while the Energy-myopic algorithm keeps low energy consumption at the price of large application delay. Additionally, tasks are processed sequentially under the Local and Sequential algorithms, so that different DAG structures have no impact on

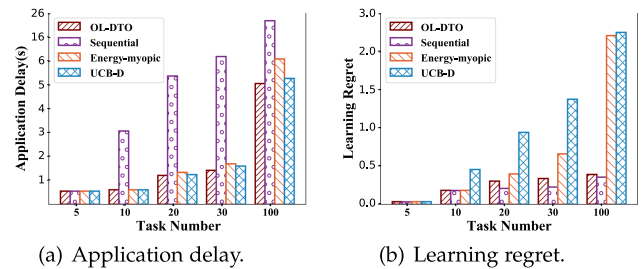


Fig. 7. Application delay and learning regret of four algorithms.

their processing. The Optimal algorithm achieves low application delay and energy consumption, but the algorithm is hard to implement in the real world.

Fig. 6 presents the application delay and local energy consumption when the UD operates both GE and FFT applications. Here, the task number changes from 40 to 220. As shown in Fig. 6, the OL-DTO algorithm achieves low application delay and energy consumption. For example, with 200 tasks, the OL-DTO algorithm reduces energy consumption by 97.79%, 86.71%, 26.50%, and 6.02% compared with the Local, Sequential, Energy-myopic, and UCB-D algorithms, respectively.

C. The Comparison of Learning-based Algorithms

Fig. 7 illustrates the application delay and learning regret of the different learning-based algorithms. With the task number increasing, all algorithms have increasing application delay and learning regret. For the Sequential algorithm, its learning times are equal to the task number, facilitating the algorithm's convergence; but the sequential offloading produces large application delay in dependent task offloading. The Energy-myopic algorithm follows the restricted single-rank energy constraint and causes prolonged application delay. The UCB-D algorithm is incapable of task dependency awareness and hence demands more learning times to realize well-balanced exploration and exploitation. It can be found that, when there are 100 tasks, the OL-DTO algorithm decreases the application delay by about 34.10%, 16.92% and 4.07% compared with the Sequential, Energy-myopic, and UCB-D algorithms, respectively; meanwhile the OL-DTO algorithm reduces the learning regret by

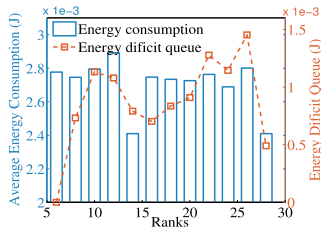
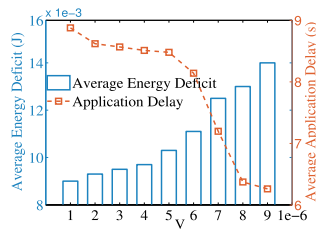


Fig. 8. Impact of energy queue.

Fig. 9. Impact of parameter V .

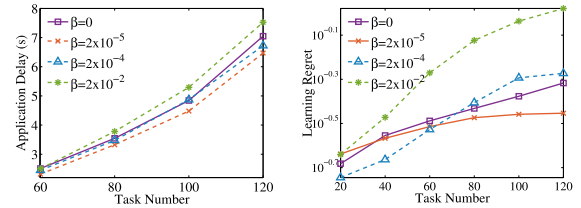
about 82.31% and 82.64% compared with the Energy-myopic and UCB-D algorithms, respectively.

D. The Impact of Different Parameters

1) *Energy Deficit Queue*: Fig. 8 depicts the impact of the energy deficit queue on the average energy consumption. A larger queue reflects more local energy violations. From the 6th to the 12th ranks, the UD consumes substantial amounts of energy and enlarges the energy deficit queue. At the 14th rank, the deficit queue adds up to 1.379 J. Guided by the queue, the UD reduces energy consumption in the following ranks. In this way, the energy deficit queue realizes online adjustments to UD's energy consumption, so that the long-term energy budget can be guaranteed in dependent task offloading.

2) *The Parameter V* : Fig. 9 displays the average energy deficit and application delay of the OL-DTO algorithm under different values of V . The results show that increasing V leads to a reduction in application delay but an increase in energy deficit. The control parameter V plays a crucial role in the Lyapunov drift-plus-penalty framework by weighting the application delay, which transforms the long-term optimization problem **P2** into an online optimization problem **P3**. A higher value of V indicates that the weighted sum of application delay and control parameter V carries more weight in **P3**, leading to lower application delay. Conversely, a lower value of V means that energy consumption holds a higher proportion in **P3**, resulting in less UD's energy consumption but more application delay. The simulation results of average application latency and energy deficiency queue under different control parameter V align with Theorem 3.

3) *The Parameter β* : Fig. 10 illustrates the impact of β on application delay and learning regret. The parameter β is a weighted factor of exploration in the online learning optimization problem. A higher value of β is beneficial in avoiding local optima but may slow down the offloading convergence.



(a) Application delay

(b) Learning regret

Fig. 10. Impact of parameter β on the application delay and learning regret.

Conversely, a lower value of β leads to less exploration, which can result in quick convergence but may overlook better edge servers and lead to poor offloading decisions. Therefore, it is crucial to strike a balance between exploration and exploitation in learning-based algorithms. According to the simulation results, we can find that the OL-DTO algorithm tends to exploit the optimal edge server without necessary exploration when $\beta = 0$. This leads to higher application delay and learning regret. Therefore, β is generally set as a positive value to explore possible better edge servers. However, excessive exploration can slow down the algorithm's convergence and impairs offloading efficiency. For example, $\beta = 2 \times 10^{-2}$ produces more application delay and learning regret than those of $\beta = 2 \times 10^{-5}$. In our simulation, we find that small explorations facilitate offloading decision-making solutions.

VI. CONCLUSION

In this article, we have studied dependent task offloading in edge computing with the unknown system-side information. We have proposed an effective online learning-based algorithm for dependent task offloading, which enables to address the unknown system-side information and is augmented by task dependency awareness. We have derived rigorous theoretical analysis to demonstrate the algorithm's performance. Extensive experimental results demonstrate that the proposed algorithm can significantly reduce application delay while satisfying the long-term UD's energy constraint. There are a few limitations that will be studied in our future work. First, we consider the impact of offloading decisions on the UD's energy consumption, while neglecting the effect of UD's transmission power. In our future work, we plan to explore the joint impact of offloading decisions and transmission energy consumption, and design energy adjustment schemes based on task offloading decisions to further reduce the UD's energy consumption under long-term energy constraints. Second, we assume that the UD's location is quasi-static and ignores the impact of mobility on task offloading. In our future work, we intend to incorporate the impact of a UD's mobility into the optimization problem and extend our research to mobile scenarios.

REFERENCES

- [1] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surv. Tut.*, vol. 19, no. 4, pp. 2322–2358, Fourth Quarter 2017.

- [2] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Joint service placement and request routing in multi-cell mobile edge computing networks," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 10–18.
- [3] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 207–215.
- [4] Z. Hu et al., "An efficient online computation offloading approach for large-scale mobile edge computing via deep reinforcement learning," *IEEE Trans. Services Comput.*, vol. 15, no. 2, pp. 669–683, Mar./Apr. 2022.
- [5] S. Ghanavati, J. Abawajy, and D. Izadi, "An energy aware task scheduling model using ant-mating optimization in fog computing environment," *IEEE Trans. Services Comput.*, vol. 15, no. 4, pp. 2007–2017, Jul./Aug. 2022.
- [6] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading dependent tasks in mobile edge computing with service caching," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 1997–2006.
- [7] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in *Proc. 25th Annu. Int. Conf. Mobile Comput. Netw.*, New York, NY, USA, 2019, doi: [10.1145/3300061.3300116](https://doi.org/10.1145/3300061.3300116).
- [8] J. Liu, J. Ren, Y. Zhang, X. Peng, Y. Zhang, and Y. Yang, "Efficient dependent task offloading for multiple applications in MEC-cloud system," *IEEE Trans. Mobile Comput.*, vol. 22, no. 4, pp. 2147–2162, Apr. 2023.
- [9] S. Sundar and B. Liang, "Offloading dependent tasks with communication delay and deadline constraint," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 37–45.
- [10] Y. Geng, Y. Yang, and G. Cao, "Energy-efficient computation offloading for multicore-based mobile devices," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 46–54.
- [11] M. Tang and V. W. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Trans. Mobile Comput.*, vol. 21, no. 6, pp. 1985–1997, Jun. 2022.
- [12] H. Wang and J. Xie, "User preference based energy-aware mobile AR system with edge computing," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 1379–1388.
- [13] Q. Li, S. Wang, A. Zhou, X. Ma, F. Yang, and A. X. Liu, "QoS driven task offloading with statistical guarantee in mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 21, no. 1, pp. 278–290, Jan. 2022.
- [14] K. Guo and T. Q. S. Quek, "On the asynchrony of computation offloading in multi-user MEC systems," *IEEE Trans. Commun.*, vol. 68, no. 12, pp. 7746–7761, Dec. 2020.
- [15] S. Tong, Y. Liu, J. Mišić, X. Chang, Z. Zhang, and C. Wang, "Joint task offloading and resource allocation for fog-based intelligent transportation systems: A UAV-enabled multi-hop collaboration paradigm," *IEEE Trans. Intell. Transp. Syst.*, to be published, doi: [10.1109/ITITS.2022.3163804](https://doi.org/10.1109/ITITS.2022.3163804).
- [16] F. Wu, S. Leng, S. Maharjan, X. Huang, and Y. Zhang, "Joint power control and computation offloading for energy-efficient mobile edge networks," *IEEE Trans. Wireless Commun.*, vol. 21, no. 6, pp. 4522–4534, Jun. 2022.
- [17] Q. Luo, C. Li, T. H. Luan, and W. Shi, "Minimizing the delay and cost of computation offloading for vehicular edge computing," *IEEE Trans. Services Comput.*, vol. 15, no. 5, pp. 2897–2909, May 2022.
- [18] Y. Du, J. Li, L. Shi, T. Liu, F. Shu, and Z. Han, "Two-tier matching game in small cell networks for mobile edge computing," *IEEE Trans. Services Comput.*, vol. 15, no. 1, pp. 254–265, Jan./Feb. 2022.
- [19] X. Wang, J. Ye, and J. C. Lui, "Decentralized task offloading in edge computing: A multi-user multi-armed bandit approach," in *Proc. IEEE Conf. Comput. Commun.*, 2022, pp. 1199–1208.
- [20] H. Wang, K. Wu, J. Wang, and G. Tang, "Rldish: Edge-assisted QoE optimization of HTTP live streaming with reinforcement learning," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 706–715.
- [21] L. Chen and J. Xu, "Task replication for vehicular cloud: Contextual combinatorial bandit with delayed feedback," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 748–756.
- [22] Y. Sun et al., "Adaptive learning-based task offloading for vehicular edge computing systems," *IEEE Trans. Veh. Technol.*, vol. 68, no. 4, pp. 3061–3074, Apr. 2019.
- [23] J. Ren et al., "An efficient two-layer task offloading scheme for MEC system with multiple services providers," in *Proc. IEEE Conf. Comput. Commun.*, 2022, pp. 1519–1528.
- [24] T. Ouyang, R. Li, X. Chen, Z. Zhou, and X. Tang, "Adaptive user-managed service placement for mobile edge computing: An online learning approach," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 1468–1476.
- [25] Z. Zhou et al., "Learning-based URLLC-aware task offloading for internet of health things," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 2, pp. 396–410, Feb. 2021.
- [26] Y. Sun, S. Zhou, and J. Xu, "EMM: Energy-aware mobility management for mobile edge computing in ultra dense networks," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2637–2646, Nov. 2017.
- [27] C. Hu and B. Li, "Distributed inference with deep learning models across heterogeneous edge devices," in *Proc. IEEE Conf. Comput. Commun.*, 2022, pp. 330–339.
- [28] Alibaba trace, Available: 2019. [Online]. Available: <https://github.com/alibaba/clusterdata>
- [29] L. Tan, Z. Kuang, L. Zhao, and A. Liu, "Energy-efficient joint task offloading and resource allocation in OFDMA-based collaborative edge computing," *IEEE Trans. Wireless Commun.*, vol. 21, no. 3, pp. 1960–1972, Mar. 2022.
- [30] Z. Nan, S. Zhou, Y. Jia, and Z. Niu, "Joint task offloading and resource allocation for vehicular edge computing with result feedback delay," *IEEE Trans. Wireless Commun.*, to be published, doi: [10.1109/TWC.2023.3244391](https://doi.org/10.1109/TWC.2023.3244391).
- [31] J. Zhang, B. Gong, M. Waqas, S. Tu, and Z. Han, "A hybrid many-objective optimization algorithm for task offloading and resource allocation in multi-server mobile edge computing networks," *IEEE Trans. Services Comput.*, to be published, doi: [10.1109/TSC.2023.3268990](https://doi.org/10.1109/TSC.2023.3268990).
- [32] W. Jiang, D. Feng, Y. Sun, G. Feng, Z. Wang, and X.-G. Xia, "Joint computation offloading and resource allocation for D2D-assisted mobile edge computing," *IEEE Trans. Services Comput.*, vol. 16, no. 3, pp. 1949–1963, May/June 2023.
- [33] G. Cui et al., "OL-EUA: Online user allocation for NOMA-based mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 22, no. 4, pp. 2295–2306, Apr. 2023.
- [34] X. An, R. Fan, H. Hu, N. Zhang, S. Atapattu, and T. A. Tsiftsis, "Joint task offloading and resource allocation for IoT edge computing with sequential task dependency," *IEEE Internet Things J.*, vol. 9, no. 23, pp. 24009–24029, Dec. 2022.
- [35] Z. Xiao et al., "Multi-objective parallel task offloading and content caching in D2D-aided MEC networks," *IEEE Trans. Mobile Comput.*, vol. 22, no. 11, pp. 6599–6615, 2023.
- [36] S. Liu et al., "Dependent task scheduling and offloading for minimizing deadline violation ratio in mobile edge computing networks," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 2, pp. 538–554, Feb. 2023.
- [37] B. Xu, Z. Kuang, J. Gao, L. Zhao, and C. Wu, "Joint offloading decision and trajectory design for UAV-enabled edge computing with task dependency," *IEEE Trans. Wireless Commun.*, vol. 22, no. 8, pp. 5043–5055, Aug. 2023.
- [38] H. Jiang, X. Dai, Z. Xiao, and A. K. Iyengar, "Joint task offloading and resource allocation for energy-constrained mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 22, no. 7, pp. 4000–4015, Jul. 2023.
- [39] J. Li et al., "Maximizing user service satisfaction for delay-sensitive IoT applications in edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 5, pp. 1199–1212, May 2022.
- [40] Y. Chen, J. Zhao, Y. Wu, J. Huang, and X. S. Shen, "QoE-aware decentralized task offloading and resource allocation for end-edge-cloud systems: A game-theoretical approach," *IEEE Trans. Mobile Comput.*, to be published, doi: [10.1109/TMC.2022.3223119](https://doi.org/10.1109/TMC.2022.3223119).
- [41] 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. Mobile Comput.*, vol. 21, no. 11, pp. 4125–4138, Nov. 2022.
- [42] Z. Ma et al., "Towards revenue-driven multi-user online task offloading in edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 5, pp. 1185–1198, May 2022.
- [43] X. Wang, Z. Ning, L. Guo, S. Guo, X. Gao, and G. Wang, "Online learning for distributed computation offloading in wireless powered mobile edge computing networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 8, pp. 1841–1855, Aug. 2022.
- [44] K. Zhang, X. Gui, D. Ren, and D. Li, "Energy-latency tradeoff for computation offloading in UAV-assisted multiaccess edge computing system," *IEEE Internet Things J.*, vol. 8, no. 8, pp. 6709–6719, Apr. 2021.
- [45] W. Zhang, R. Yadav, Y.-C. Tian, S. K. S. Tyagi, I. A. Elgendy, and O. Kaiwartya, "Two-phase industrial manufacturing service management for energy efficiency of data centers," *IEEE Trans. Ind. Inform.*, vol. 18, no. 11, pp. 7525–7536, Nov. 2022.

- [46] R. Yadav, W. Zhang, O. Kaiwartya, H. Song, and S. Yu, "Energy-latency tradeoff for dynamic computation offloading in vehicular fog computing," *IEEE Trans. Veh. Technol.*, vol. 69, no. 12, pp. 14198–14211, Dec. 2020.
- [47] R. Zhang, P. Cheng, Z. Chen, S. Liu, B. Vucetic, and Y. Li, "Calibrated bandit learning for decentralized task offloading in ultra-dense networks," *IEEE Trans. Commun.*, vol. 70, no. 4, pp. 2547–2560, Apr. 2022.
- [48] B. Zhu, K. Chi, J. Liu, K. Yu, and S. Mumtaz, "Efficient offloading for minimizing task computation delay of NOMA-based multiaccess edge computing," *IEEE Trans. Commun.*, vol. 70, no. 5, pp. 3186–3203, May 2022.
- [49] T. M. Ho and K.-K. Nguyen, "Joint server selection, cooperative offloading and handover in multi-access edge computing wireless network: A deep reinforcement learning approach," *IEEE Trans. Mobile Comput.*, vol. 21, no. 7, pp. 2421–2435, Jul. 2022.
- [50] R. Lu et al., "Auction-based cluster federated learning in mobile edge computing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 4, pp. 1145–1158, Apr. 2023.
- [51] R. Yadav et al., "Smart healthcare: RL-based task offloading scheme for edge-enable sensor networks," *IEEE Sensors J.*, vol. 21, no. 22, pp. 24910–24918, Nov. 2021.
- [52] A. Samy, I. A. Elgandy, H. Yu, W. Zhang, and H. Zhang, "Secure task offloading in blockchain-enabled mobile edge computing with deep reinforcement learning," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 4, pp. 4872–4887, Dec. 2022.
- [53] X. Dai et al., "A learning-based approach for vehicle-to-vehicle computation offloading," *IEEE Internet Things J.*, vol. 10, no. 8, pp. 7244–7258, Apr. 2023.
- [54] M. Goudarzi, H. Wu, M. Palaniswami, and R. Buyya, "An application placement technique for concurrent IoT applications in edge and fog computing environments," *IEEE Trans. Mobile Comput.*, vol. 20, no. 4, pp. 1298–1311, Apr. 2021.
- [55] L. Pan, L. Wang, S. Chen, and F. Liu, "Retention-aware container caching for serverless edge computing," in *Proc. IEEE Conf. Comput. Commun.*, 2022, pp. 1069–1078.
- [56] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.
- [57] M. Neel, *Stochastic Network Optimization with Application to Communication and Queueing Systems*. San Rafael, CA, USA: Morgan Claypool, 2012.
- [58] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Mach. Learn.*, vol. 47, no. 2/3, pp. 235–256, 2002.
- [59] N. Eshraghi and B. Liang, "Joint offloading decision and resource allocation with uncertain task computing requirement," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 1414–1422.
- [60] J. Kwak, Y. Kim, J. Lee, and S. Chong, "DREAM: Dynamic resource and task allocation for energy minimization in mobile cloud systems," *IEEE J. Sel. Areas Commun.*, vol. 33, no. 12, pp. 2510–2523, Dec. 2015.
- [61] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1619–1632, Aug. 2018.
- [62] C. Wang, S. Zhang, Y. Chen, Z. Qian, J. Wu, and M. Xiao, "Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics," in *Proc. IEEE Conf. Comput. Commun.*, 2020, pp. 257–266.



Xingxia Dai received the BS degree in communication engineering from Xiangtan University, Xiangtan, China, in 2018. She is currently working toward the PhD degree in computer science and technology with the Hunan University, Changsha, China. Her current research interests include internet of vehicles and mobile edge computing.



Zhu Xiao (Senior Member, IEEE) received the MS and PhD degrees in communication and information system from Xidian University, China, in 2007 and 2009, respectively. From 2010 to 2012, he was a research fellow with the Department of Computer Science and Technology, University of Bedfordshire, U.K. He is currently a full professor with the College of Computer Science and Electronic Engineering, Hunan University, China. His research interests include mobile communications, wireless localization, Internet of Vehicles, and trajectory data mining.



Hongbo Jiang (Senior Member, IEEE) received the PhD degree from Case Western Reserve University, in 2008. He is currently a full professor with the College of Computer Science and Electronic Engineering, Hunan University. He was a professor with the Huazhong University of Science and Technology, China. His research concerns computer networking, especially algorithms and protocols for wireless and mobile networks. He was the editor of *IEEE/ACM Transactions on Networking*, the associate editor for *IEEE Transactions on Mobile Computing*, and the associate technical editor for *IEEE Communications Magazine*. He is an elected member of Academia Europaea, fellow of IET, fellow of BCS, and fellow of AAlA.



Ming Lei received the MS degree in 2000. He is currently a member of the Communist Party of China, a senior engineer, winner of China Unicom's "Outstanding Party Member" and "Hunan Provincial May 1st Labor Medal", an outstanding management talent for the digital transformation of the group company, an exceptional manager of Hunan Unicom, and an advanced individual. His current research interests include wireless communication and mobile communication.



Geyong Min (Member, IEEE) received the BSc degree in computer science from the Huazhong University of Science and Technology, China, in 1995 and the PhD degree in computing science from the University of Glasgow, U.K., in 2003. He is a professor of high performance computing and networking in the Department of Computer Science within the College of Engineering, Mathematics and Physical Sciences with the University of Exeter, United Kingdom. His research interests include future internet, computer networks, wireless communications, multimedia systems, information security, high performance computing, ubiquitous computing, modelling and performance engineering.



Jiangchuan Liu (Fellow, IEEE) received the BEng degree (cum laude) from Tsinghua University, Beijing, China, in 1999, and the PhD degree from the Hong Kong University of Science and Technology, in 2003, both in computer science. He is a university professor in the School of Computing Science, Simon Fraser University, British Columbia, Canada. He is a fellow of the Canadian Academy of Engineering and an NSERC E.W.R. Steacie Memorial fellow. He was an EMC Endowed visiting chair professor of Tsinghua University (2013–2016). In the past, he

worked as an assistant professor with the Chinese University of Hong Kong and as a research fellow with Microsoft Research Asia. He is a corecipient of the inaugural Test of Time Paper Award of IEEE INFOCOM (2015), ACM SIGMM TOMCCAP Nicolas D. Georganas Best Paper Award (2013), and ACM Multimedia Best Paper Award (2012). His research interests include multimedia systems and networks, cloud and edge computing, social networking, online gaming, and Internet of things/RFID/backscatter. He has served on the editorial boards of *IEEE/ACM Transactions on Networking*, *IEEE Transactions on Big Data*, *IEEE Transactions on Multimedia*, *IEEE Communications Surveys and Tutorials*, and *IEEE Internet of Things Journal*. He is a steering committee member of *IEEE Transactions on Mobile Computing and Steering Committee* Chair of IEEE/ACM IWQoS (2015–2017). He is TPC Co-Chair of IEEE INFOCOM'2021.



Shahram Dustdar (Fellow, IEEE) received the PhD degree in business informatics from the University of Linz, Austria, in 1992. He is currently a full professor of computer science (informatics) with a focus on internet technologies heading the Distributed Systems Group, TU Wien, Wien, Austria. He has been the Chairman of the Informatics Section of the Academia Europaea, since 2016. Since 2013, he has been the member of Academia Europaea: The Academy of Europe, Informatics Section, Section Committee of Informatics of the Academia Europaea, since 2015,

and IEEE Conference Activities Committee (CAC), since 2016.. He was a recipient of the ACM Distinguished Scientist Award, in 2009 and the IBM Faculty Award, in 2012. He is an associate editor of the *IEEE Transactions on Services Computing*, *ACM Transactions on the Web*, and *ACM Transactions on Internet Technology*. He is on the editorial board of IEEE.