Real-Time Offloading for Dependent and Parallel Tasks in Cloud-Edge Environments Using Deep Reinforcement Learning

Xing Chen, *Member, IEEE*, Shengxi Hu, Chujia Yu, Zheyi Chen, *Member, IEEE*, and Geyong Min, *Member, IEEE*

***Abstract*—As an effective technique to relieve the problem of resource constraints on mobile devices (MDs), the computation offloading utilizes powerful cloud and edge resources to process the computation-intensive tasks of mobile applications uploaded from MDs. In cloud-edge computing, the resources (e.g., cloud and edge servers) that can be accessed by mobile applications may change dynamically. Meanwhile, the parallel tasks in mobile applications may lead to the huge solution space of offloading decisions. Therefore, it is challenging to determine proper offloading plans in response to such high dynamics and complexity in cloud-edge environments. The existing studies often preset the priority of parallel tasks to simplify the solution space of offloading decisions, and thus the proper offloading plans cannot be found in many cases. To address this challenge, we propose a novel real-time and Dependency-aware task Offloading method with Deep Q-networks (DODQ) in cloud-edge computing. In DODQ, mobile applications are first modeled as Directed Acyclic Graphs (DAGs). Next, the Deep Q-Networks (DQN) is customized to train the decision-making model of task offloading, aiming to quickly complete the decision-making process and generate new offloading plans when the environments change, which considers the parallelism of tasks without presetting the task priority when scheduling tasks. Simulation results show that the DODQ can well adapt to different environments and efficiently make offloading decisions. Moreover, the DODQ outperforms the state-of-art methods and quickly reaches the optimal/near-optimal performance.**

***Index Terms*—Cloud-edge computing, deep reinforcement learning, dependent and parallel tasks, real-time offloading.**

Manuscript received 19 December 2022; revised 3 July 2023; accepted 29 December 2023. Date of publication 3 January 2024; date of current version 23 January 2024. This work was supported in part by the National Natural Science Foundation of China under Grant 62072108, in part by the Natural Science Foundation of Fujian Province for Distinguished Young Scholars under Grant 2020J06014, in part by Central Funds Guiding the Local Science and Technology Development under Grant 2022L3004, in part by Fujian Province Technology and Economy Integration Service Platform under Grant 2023XRH001, and in part by Fuzhou-Xiamen-Quanzhou National Independent Innovation Demonstration Zone Collaborative Innovation Platform under Grant 2022FX5. Recommended for acceptance by M. Fazio. *(Corresponding authors: Zheyi Chen; Geyong Min.)*

Xing Chen, Shengxi Hu, Chujia Yu, and Zheyi Chen are with the College of Computer and Data Science, Fuzhou University, Fuzhou 350116, China, also with the Engineering Research Center of Big Data Intelligence, Ministry of Education, Fuzhou 350002, China, and also with the Fujian Key Laboratory of Network Computing and Intelligent Information Processing, Fuzhou University, Fuzhou 350116, China (e-mail: chenxing@fzu.edu.cn; husx0913@163.com; 221027123@fzu.edu.cn; z.chen@fzu.edu.cn).

Geyong Min is with the Department of Computer Science, Faculty of Environment, Science and Economy, University of Exeter, EX4 4QF Exeter, U.K. (e-mail: g.min@exeter.ac.uk).

Digital Object Identifier 10.1109/TPDS.2023.3349177

I. Introduction

With the rapid development of intelligent technology, the increasing number of computation-intensive applications, such as autonomous driving [1], face recognition [2], and augmented reality [3], have emerged to better meet the growing user demands. Meanwhile, the operating platforms of these applications have extended from laptops and smartphones to extensive mobile devices (MDs) [4], such as wearable devices, vehicles, and drones. However, due to the constraints on size and weight, MDs reveal obvious limitations on processing capability and battery capacity [5], and thus MDs may be unable to efficiently handle the computation-intensive tasks of mobile applications.

To relieve the issue of resource constraints on MDs, the computation offloading technique [6], [7], [8] has been proposed to offload the computation-intensive tasks from MDs to remote servers for execution and thus expand local computing capability by utilizing powerful remote resources. Mobile Cloud Computing (MCC) integrates the computing capability of the cloud into mobile networks, which enables mobile users to access cloud computing services via the core network [9]. In MCC, the computation-intensive tasks of mobile applications can be uploaded to cloud servers for execution, but there is commonly a long distance between the remote cloud and MDs, which may result in excessive network latency. To address this problem, the emerging Mobile Edge Computing (MEC) considers deploying servers at the network edge that are close to MDs [10]. In MEC, different numbers of computing resources are distributed on MDs and edge servers, and the resources that mobile applications can access are dynamically changing as the locations of MDs change. The offloading plans determine the computing platforms of executing tasks, but there is no offloading plan that is universal. When the environments change, the offloading plans need to be adjusted to ensure the good performance of mobile applications. Therefore, the decision-makings of task offloading are expected to be made at runtime. Considering the dynamic environments with high mobility of MDs, the selection of offloading plans is a problem with the combinatorial

explosion [11]. Thus, it is extremely hard to efficiently make offloading plans to satisfy the real-time service demands of mobile applications.

Recently, there have been some studies focusing on resolving the offloading problem in cloud or edge environments. On one hand, the studies [12], [13], [14], [15], [16] adopted heuristic algorithms to explore suitable offloading plans. However, the runtime environments of mobile applications may often change, which leads to the problem of the combinatorial explosion when making offloading decisions. Therefore, they might consume tens or even hundreds of seconds to find a suitable offloading plan, and this low efficiency cannot well meet the real-time service requirements of mobile applications. On the other hand, the studies [17], [18], [19], [20], [21] introduced Reinforcement Learning (RL) to solve the offloading problem in cloud or edge environments. However, these RL-based offloading models commonly preset the order of task scheduling but ignore the order arbitrariness of task scheduling when scheduling parallel tasks, which limited the solution space of searching for better offloading plans. Moreover, they only consider the offloading problem in edge environments but do not well study a hierarchical cloud-edge framework to further enhance the offloading performance.

To address the above problems, we propose a real-time and Dependency-aware task Offloading method with Deep Q-networks (DODQ) to quickly and adaptively find the optimal offloading plan in complex and dynamic cloud-edge environments. The main contributions of this paper are summarized as follows.

- The task offloading in cloud-edge computing is formulated as an optimization problem, aiming to minimize the response time of mobile applications that are modeled as Directed Acyclic Graphs (DAGs).
- A Deep Q-Networks (DQN) algorithm is customized to train the decision-making model of task offloading that considers the parallelism of tasks without presetting the task priority when scheduling tasks, which can well adapt to various cloud-edge environments and quickly generate the optimal offloading plan.
- Extensive simulation results demonstrate the excellent adaptability of the proposed DODQ to different cloud-edge environments, which takes only milliseconds to generate offloading plans. Moreover, the DODQ outperforms the state-of-art methods and quickly reaches the optimal/near-optimal response time.

The rest of this paper is organized as follows. In Section II the related work is reviewed. Section III formulates the problem of task offloading in cloud-edge environments. In Section IV, the proposed DODQ is presented in detail. Section V analyzes the evaluation results. In Section VI, we discuss the problem complexity and the evaluation in real-world environments. Section VII makes conclusion for this paper.

II. Related Work

Commonly, MDs are constrained on their limited storage space, battery life, and computing capacity [22]. To improve the performance of mobile applications, the computation-intensive tasks of mobile applications can be offloaded from MDs to the cloud or edge servers for execution. In this section, the related work on the offloading problem in cloud or edge environments is reviewed and analyzed.

Most of the existing studies on the offloading problem adopt heuristic algorithms. For example, Zhu et al. [12] designed an iterated heuristic framework to schedule linearly-dependent tasks in a hybrid-cloud-based workflow management system. Chen et al. [13] developed a multi-objective ant colony system based on co-evolutionary multiple populations, to optimize execution time and execution cost of cloud workflow scheduling. Kaur et al. [14] proposed an augmented shuffled frog leaping algorithm to minimize the processing time of the individual tasks in a work flow and the transfer time between dependent tasks. Jia et al. [15] designed an ant colony optimization method to tackle the problem of cloud workflow scheduling. Xie et al. [16] proposed a directional and non-local-convergent particle swarm optimization to reduce the makespan and cost of workflow scheduling in a cloud-edge environment. However, the above studies target the scenarios with the preset the order of task scheduling when scheduling parallel tasks. As for the studies with the consideration of the arbitrary order of task scheduling when scheduling parallel tasks, Cheng et al. [23] developed an optimization framework for the offloading problem by jointly considering task assignment and task scheduling. Wang et al. [24] proposed a hybrid particle swarm optimization algorithm for the problem of cloud workflow scheduling, which selected the most appropriate VM type for each task and determined the best task scheduling order. Tong et al. [25] integrated a migration strategy with particle swarm optimization to solve the problem of scheduling DAG tasks in a cloud environment. Guo et al. [26] designed a discrete particle swarm optimization technique with two-point crossover and single-point mutation of the genetic algorithm, aiming to reduce the execution cost of scientific workflows in the cloud. However, the selection of offloading plans faces the problem of the combinatorial explosion, while these heuristic algorithms cannot address this problem in a short time. This low efficiency cannot well meet the real-time service requirements of mobile applications.

In recent, advanced RL-based methods have also been applied to handle the offloading problem. For example, Cui et al. [27] proposed an RL-based algorithm to handle the problem of scheduling DAG workflows in cloud environments. Wu et al. [28] designed a Q-learning approach to address the multi-objective optimization problem of cloud workflow scheduling. Nascimento et al. [29] developed an RL-based activation algorithm to optimize the scheduling issue in the cloud with the consideration of previous executions. Pan et al. [30] proposed a Q-learning approach to optimize the execution time and energy consumption of mobile applications in MEC. However, the Q-learning cannot work well in large-scale problems because it stores all state-action pairs in a Q-table. Moreover, when the runtime environments change, the Q-learning based decision-making model needs to be retrained for fitting in a new environment. To address this issue, Qian et al. [31] studied the offloading problem in dynamic NOMA scenarios and proposed

an online DRL-based algorithm to find suitable offloading plans. Chen et al. [32] designed a DQN-based ofoading method to optimize policies without prior knowledge of dynamic edge environments. In these two studies, they did not fully consider the dependent relationship between tasks. In contrast, Wang et al. [17] designed a DRL-based framework to efficiently learn the offloading policy for dependent tasks with the combination of sequence-to-sequence neural networks. Yan et al. [18] utilized the DRL to learn the optimal mapping between input states and binary offloading decisions in a single-user MEC system with a task graph. Zhu et al. [19] proposed a DQN-based offloading algorithm to reduce the energy consumption and completion time of executing service workflows. Peng et al. [20] designed a DQN-based cloud resource management framework to solve the scheduling problem in cloud environments. Song et al. [21] developed an improved multi-objective RL algorithm to address the problem of offloading dependent tasks. However, these studies commonly preset the order of task scheduling when scheduling parallel tasks, which limited the solution space of searching for better offloading plans. Moreover, the above studies only consider the offloading problem in edge or cloud environments but do not well study a hierarchical cloud-edge framework to further improve the offloading performance.

In general, most of the existing work focused on the offloading problem in cloud or edge environments, and they revealed the limitations on real-time and adaptability. Different from the related work, in this paper, we first model the mobile applications as DAGs that consist of many tasks with certain dependencies. Next, we consider the order arbitrariness of task scheduling without presetting the task priority when scheduling parallel tasks during the offloading process and then realize efficient and adaptive task offloading in dynamic cloud-edge environments.

III. System Model and Problem Formulation

A computation-intensive mobile application consists of many tasks with certain dependencies [33], and these tasks can be executed on local MDs or offloaded to remote cloud and edge servers for execution. However, unreasonable offloading plans may cause excessive response time on mobile applications. On one hand, due to the limited computing capability of MDs, long processing latency may happen if the computation-intensive tasks of mobile applications are executed locally. On the other hand, serious transmission latency may occur by offloading the tasks of mobile applications to remote servers if there exist frequent data exchanges between different tasks.

A. Network Model

We consider a system model that consists of an MD, an edge server, and a cloud server, as shown in Fig. 1, and the main symbols used in the proposed model are listed in Table I. Specifically, $K = \{MD, ES, CS\}$ denotes the set of computing nodes, where MD, ES, and CS represent the mobile device, edge server, and cloud server, respectively. The computing capability of a node is denoted as f_k ($k \in K$), and the data transmission rate between the nodes k and l is denoted as $v_{k,l}$ ($k, l \in K$).

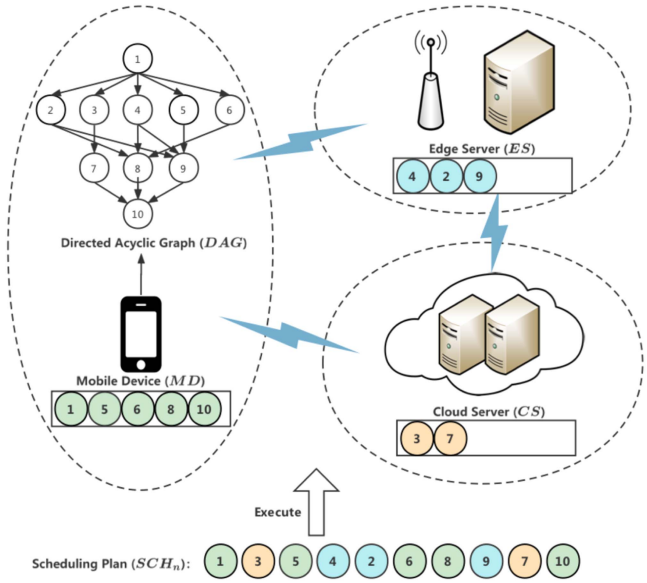


Fig. 1. Proposed system model of task offloading.

TABLE I
Main Symbols Used in the Proposed Model

Symbol	Description
K	Set of computing nodes including MD, ES, and CS
f_k	Computing capability of k ($k \in K$)
$v_{k,l}$	Data transmission rate between k and l ($k, l \in K$)
N	Set of tasks in an application, where $N = \{1, 2, ..., n\}$
c_i	Calculation amount of the task i ($i \in N$)
E	Set of dataflow edges, where $e_{i,j} \in E$
$d_{i,j}$	Size of data transmission from the task i to j
$pre(i)$	Set of predecessor tasks of the task i
SCH_n	Scheduling plan of an application
DEP_t	Task scheduling of an application at time step t
l_{task}	Set of tasks that must be executed on MD
T_t	Latency of an application at time step t
T_{resp}	Response time of an application
D_t	Set of completed tasks at time step t
$t_{end}(i)$	End time of the task i
$t_{exe}(i)$	Execution time of the task i
$t_{start}(i)$	Start time of the task i
$t_{tran}(i, j)$	Data transmission time from the task i to j
$t_{avl}(k)$	The time when the node k becomes available

B. Task Model

A mobile application is modeled as a DAG, denoted by $G = (N, E)$. $N = \{1, 2, \ldots, n\}$ is the set of tasks, and it is considered that some tasks in the DAG can be executed in parallel. c_i ($i \in N$) represents the calculation amount of a task. E is the set of dataflow edges. For a dataflow edge $e_{i,j}$ ($e_{i,j} \in E$), the task i is the predecessor of the task j and the task j is the successor of the task i. Specially, $d_{i,j}$ indicates the size of data transmission from the task i to the task j. Due to the task dependency, a task can only be executed after receiving the processing results of all its predecessors, where $pre(i)$ indicates the set of predecessor tasks of the task i. Taking the DAG in Fig. 1 as an example, the set of predecessor tasks of the task 10 is $pre(10) = \{7, 8, 9\}$,

and thus the task 10 must be waiting for the data from the tasks 7, 8, and 9 before it can be executed.

C. Task Scheduling Plan

For an application with n tasks, the scheduling plan is defined as $SCH_n = (sch_1, sch_2, \ldots, sch_n)$, where sch_t indicates the scheduling decision at time step t. Specifically, $sch_t = task_i^k$ indicates that the task i is scheduled to the computing node k. As shown in Fig. 1, when the scheduling decision follows: (1) the task 1 is scheduled to MD, (2) the task 3 is scheduled to CS, (3) the task 5 is scheduled to MD,..., (10) the task 10 is scheduled to MD, $SCH = (task_1^{MD}, task_3^{CS}, task_5^{MD}, \ldots, task_{10}^{MD})$. To make better clarification, the task scheduling of an application at time step t is defined as $DEP_t = (dep_t(1), dep_t(2), \ldots, dep_t(n))$, where $dep_t(i) \in \{0, 1, 2, 3\}$ indicates the scheduling plan of the task i. Specifically, $dep_t(i) = 0$ indicates that the task i has not been scheduled at time step t, and $dep_t(i) = 1$, 2, or 3 indicates that the task i has been scheduled at time step t and executed on MD, ES, or CS. Based on the above scheduling plan, the scheduling process of an application can be described as $DEP_0 = (0, 0, 0, \ldots, 0), DEP_1 = (1, 0, 0, \ldots, 0), DEP_2 = (1, 0, 3, \ldots, 0), \ldots, DEP_{10} = (1, 3, 2, \ldots, 1)$. According to the scheduling decisions at different time steps, the final scheduling plan SCH_n can be obtained and the tasks of an application will be executed via a parallel manner based on SCH_n. Moreover, T_t indicates the latency of an application at time step t, where the detailed calculation process of T_t will be given in Section III-D. Thus, the response time of an application is defined as T_{resp}, which indicates the latency after completing the last task (at time step $t = n$), and it is defined as

$$T_{resp} = T_n. \tag{1}$$

In real-world scenarios, some tasks must be executed on MD such as user-interaction tasks and device I/O tasks [34]. Therefore, the set of tasks that must be executed on MD is defined as l_{task}, where $DEP_t(i) \in 0, 1$ $(i \in l_{task})$.

D. Calculation Process of T_t

As aforementioned, T_t indicates the latency of an application at time step t. Since tasks are executed in parallel on different nodes, T_t indicates the maximum end time of all completed tasks at time step t. Thus, T_t can be defined as

$$T_t = \max_{i \in D_t} \{t_{end}(i)\}, \tag{2}$$

where D_t is the set of completed tasks at time step t. $t_{end}(i)$ is the end time of the task i, which is defined as

$$t_{end}(i) = t_{exe}(i) + t_{start}(i), \tag{3}$$

where $t_{exe}(i)$ and $t_{start}(i)$ are the execution and start time of the task i, respectively.

On one hand, the execution time of the task i depends on the computing capability of the node k and the calculation amount of the task i, which is defined as

$$t_{exe}(i) = \frac{c_i}{f_k}. \tag{4}$$

On the other hand, the start time of the task i depends on two aspects. First, the task i should receive the execution results of all its predecessor tasks. Second, the node that executes the task i should be available. Therefore, the start time of the task i is defined as

$$t_{start}(i) = \max \left\{ \max_{j \in pre(i)} \{t_{end}(j) + t_{tran}(i, j)\}, t_{avl}(k) \right\}, \tag{5}$$

where $t_{avl}(k)$ is the time when the node k becomes available. $t_{tran}(i, j)$ is the data transmission time from the task i to j, which is defined as

$$t_{tran}(i, j) = \frac{d_{i,j}}{v_{k,l}}. \tag{6}$$

If the task i and one of its predecessor tasks j are scheduled to nodes k and l respectively, the data transmission time between nodes should be considered. Otherwise, there is no data transmission time as the data transmission can be achieved through shared memory.

Algorithm 1 describes the calculation process of T_t according to the above constraints and definitions, where T_t can be calculated according to the task scheduling decision sch_t at time step t. If it is the first decision, D_0 and $t_{avl}(k)$ will be initialized; otherwise, D_{t-1} and $t_{avl}(k)$ will be got from the former time step (Lines 6∼10). First, the current task and the execution location are got according to sch_t (Line 11). Next, all the predecessor tasks of $task\,i$ are traversed to obtain the data transmission time between tasks (Lines 12∼18). Then, $t_{start}(i)$, $t_{exe}(i)$, and $t_{end}(i)$ are calculated according to (3) ∼ (5) (Line 20). Next, $t_{avl}(k)$ and D_t are updated (Lines 21∼22). Finally, T_t is calculate and output according to (2) (Line 23).

E. Problem Formulation

Different offloading plans may cause various values of T_n. To make better task offloading with lower value of T_{resp}, the objective function is defined as

$$\textbf{Minimize}\ T_{resp}. \tag{7}$$

Based on the above definitions, the runtime environment is denoted as a 2-tuple $\langle F, V \rangle$. As shown in Table II, $F = (f_{MD}, f_{ES}, f_{CS})$ contains the computing capabilities of all nodes (i.e., MD, edge server, and cloud server), and $V = (v_{MD,ES}, v_{MD,CS}, v_{ES,CS})$ contains the data transmission rates between these nodes. Moreover, T_{resp} is regarded as the performance metric of SCH_n.

Considering the high mobility of MDs, the runtime environments of mobile applications may change frequently, and thus the selection of offloading plans would be a problem with the combinatorial explosion. It may take tens or even hundreds of seconds to find a suitable offloading plan by using some traditional heuristic algorithms, which can not well meet the real-time service requirements of mobile applications. In response to this problem, Deep Reinforcement Learning (DRL) will be used to implement adaptive and fast task offloading in dynamic cloud-edge environments.

Algorithm 1: Calculation Process of T_t.

Input: sch_t.
Output: T_t.
Declaration
  $task\,i$: the task of sch_t.
  $excute\_location\,k$: the execution location in sch_t.
if $t = 1$ **then**
  Initialize $D_0 = \{\}$ and $t_{avl}(k) = 0$;
else
  Get D_{t-1} and $t_{avl}(k)$;
end
Get $task\,i$ and $excute\_location\,k$;
for each $pre\_taskj \in pre(i)$ **do**
  Get $excute\_location\,l$ of $task\,k$ from $DEP_t[j]$;
  if $k == l$ **then**
    $t_{tran}(i,j) = 0$;
  else
    $t_{tran}(i,j) = \frac{d_{i,j}}{v_{k,l}}$;
  end
end
Calculate $t_{start}(i)$, $t_{exe}(i)$, and $t_{end}(i)$ according to (3) $\sim$ (5);
Update $t_{avl}(k) = t_{end}(i)$;
Update D_t: add $task\,i$ to D_{t-1};
Calculate and output T_t according to (2).

TABLE II
RUNTIME ENVIRONMENT AND OFFLOADING PLAN WITH PERFORMANCE METRIC

Runtime environment	
F	f_{MD}, f_{ES}, f_{CS}
V	$v_{MD,ES}, v_{MD,CS}, v_{ES,CS}$
Offloading plan with performance metric	
SCH_n	$sch_1, sch_2, ..., sch_n$
T_{resp}	T_n

IV. Adaptive and Fast Task Offloading With DQN in Cloud-Edge Computing

In this section, we present the proposed real-time and Dependency-aware task Offloading method with Deep Q-networks (DODQ), which can be used to quickly and adaptively find the optimal offloading plan in complex and dynamic cloud-edge environments. The main steps of the DODQ are given as follows.

Step 1: The Deep Q-Networks (DQN) algorithm [35] is used to train the Q-value prediction model of offloading operations, where the system state consists of F, V, DEP_t in the runtime environment. The scheduling operation is to execute a task on MD or offload it to ES or CS. When the scheduling operation is completed, the corresponding reward will be calculated. Through training, the Q-value prediction model can accurately evaluate the Q-values of different offloading operations under various runtime cloud-edge environments.

Step 2: The offloading operations are selected based on their predicted Q-values. By the iterative feedback-control process, the optimal offloading plan can be gradually found at the runtime decision-making process, which determines the execution locations of each task. Finally, the tasks of an application will be executed via a parallel manner according to the above plan.

A. Q-Value Prediction of Offloading Operations

As an advanced DRL algorithm, the DQN integrates the Q-learning [36] with DNNs [37]. Specifically, the DQN fits the Q-value function by using DNNs rather than performs cost-consuming searches in a large Q-value table. Hence, the DQN is applied to evaluate the Q-values of offloading operations under different system states.

Table III shows an example of the offloading process with 10 tasks. First, the task 1 is executed on MD, and the corresponding $DEP_1 = (1,0,0,0,0,\ldots,0)$ and $T_1 = 0.161$s. Next, the task 3 is offloaded to CS for execution, and the corresponding $DEP_2 = (1,0,3,0,0,\ldots,0)$ and $T_2 = 0.265$s. Similarly, the execution locations of each task are determined in turn. Finally, when $DEP_{10} = (1,2,3,2,1,\ldots,1)$, all the tasks have been executed and the response time of an application can be calculated (i.e., $T_{resp} = T_{10} = 0.515$s). It is noted that the tasks in the DAG are executed in parallel (e.g., $T_3 = T_4 = 0.282\,s$), and thus the latency of an application is equal to the longest time it takes to complete these dependent tasks in the DAG.

By training the DQN agent, the offloading operation with the highest Q-value can be found under different system states, where the DQN algorithm is used to control the Q-value evaluation process. The DRL problems are usually modeled as Markov decision processes (MDPs), aiming to maximize the accumulative rewards. Specifically, an MDP is denoted as a 4-tuple $\langle S, A, P, R\rangle$, where S, A, P, and R are the state space, action space, state-transition function, and reward function, respectively. Based on the problem formulation of task offloading, they are defined as follows.

State space: The state space is defined as S, where $s_t \in S$ represents the state at time step t. Specifically, s is defined as a 3-tuple $\langle F, V, DEP_t\rangle$, which consists of the computing capabilities of different nodes, the data transmission rates between these nodes, and the task scheduling at time step t. Taking Table III as an example, when $t = 3$, $s_t = (F, V, (1,0,3,0,1,\ldots,0))$.

Action space: The action space is defined as $A = \{task_1^{MD}, task_1^{ES}, task_1^{CS}, \ldots, task_n^{MD}, task_n^{ES}, task_n^{CS}\}$, where an action a_t ($a_t \in A$) indicates the scheduling decision at time step t.

State-transition function: The state-transition function is defined as $P(s_t, a_t)$, which returns the next state after executing a_t at s_t. Taking Table III as an example, after executing $a_3 = task_2^{ES}$ at $s_3 = (F, V, (1,0,3,0,1,\ldots,0))$, the state is transferred to $s_4 = P(s_3, a_3) = (F, V, (1,0,3,2,1,\ldots,0))$.

Reward function: The reward function is used to guide the DQN agent to learn the optimal offloading plan for minimizing the response time, and thus it is defined as

$$r_t = R(s_t, a_t) = T_t - T_{t+1}, \tag{8}$$

TABLE III
EXAMPLE OF THE OFFLOADING PROCESS WITH 10 TASKS

t	s_t	a_t	SCH_t	T_t
0	$(F, V, (0, 0, 0, 0, 0, ..., 0))$	$task_1^{MD}$	–	0s
1	$(F, V, (1, 0, 0, 0, 0, ..., 0))$	$task_3^{CS}$	$(task_1^{MD})$	0.161s
2	$(F, V, (1, 0, 3, 0, 0, ..., 0))$	$task_5^{MD}$	$(task_1^{MD}, task_3^{CS})$	0.265s
3	$(F, V, (1, 0, 3, 0, 1, ..., 0))$	$task_4^{ES}$	$(task_1^{MD}, task_3^{CS}, task_5^{MD})$	0.282s
4	$(F, V, (1, 0, 3, 2, 1, ..., 0))$	$task_2^{ES}$	$(task_1^{MD}, task_3^{CS}, task_5^{MD}, task_4^{ES})$	0.282s
...	...	...	...	...
t	(F, V, DEP_t)	a_t	$(sch_1, sch_2, sch_3, sch_4, sch_5, ..., sch_t)$	T_t
...	...	...	...	...
10	$(F, V, (1, 2, 3, 2, 1, ..., 1))$	–	$(task_1^{MD}, task_3^{CS}, task_5^{MD}, task_4^{ES}, task_2^{ES}, ..., task_{10}^{MD})$	0.515s

Algorithm 2: Selecting Actions That Meet Constraints.

1 **Input:** DEP_t.
2 **Output:** The valid action space of DEP_t.
3 **Declaration**
4 $task\_no$: the task selected by a.
5 $excute\_location$: the execution location decided by a.
6 l_task: the task set executed on the mobile device.
7 **for** each $a \in A$ **do**
8 $a = True$; Get $task\_no$ and $excute\_location$;
9 **if** $task\_no \in l\_task$ and $excute\_location \neq MD$ or $task\_no$ has been scheduled in DEP_t **then**
10 $a = False$;
11 continue; // Not meet the constraints (1)(2)
12 **end**
13 **for** each $pre\_task \in pre(task\_no)$ **do**
14 **if** $task\_no$ is not scheduled in DEP_t **then**
15 $a = False$;
16 break; // Not meet the constraint (3)
17 **end**
18 **end**
19 **end**
20 **Return** the valid action space of DEP_t.

where r_t indicates the reward after executing a_t at s_t, T_t is the accumulated latency at time step t, T_{t+1} is the accumulated latency at time step $t+1$ after executing a_t at s_t. Therefore, $T_t - T_{t+1}$ is a negative value. Taking Table III as an example, $r_3 = R(s_3, a_3) = T_3 - T_4 = 0.282 - 0.282 = 0$. To maximize the accumulative rewards, the response time should be minimized.

It is noted that it is not allowed to execute certain scheduling operations under some conditions. Specifically, a scheduling operation should meet the following constraints: (1) A task cannot be offloaded to the edge/cloud server if it must be executed on the mobile device; (2) A task cannot be scheduled again if it has been scheduled and executed; (3) Due to the task dependency, a task can only be executed after receiving the results of all its predecessors. Therefore, Algorithm 2 is designed to achieve the above objectives. Specifically, we utilize DEP_t to obtain the set of actions that meet the constraints, which ensures that the DRL can select effective actions during the learning process, to reduce the complexity of the algorithm and accelerate the convergence.

Fig. 2 shows the framework of DQN for task offloading in the cloud-edge environment. First, the DQN agent observes the state s_t in the runtime cloud-edge environment and selects an action a_t by using the ϵ-greedy strategy. Next, the DQN agent receives the reward r and the next state s_{t+1}. Meanwhile, (s_t, a_t, r_t, s_{t+1}) of each step will be stored in the replay memory. Once reaching the predefined threshold of memory capacity, the parameters of DNNs will be updated, and the corresponding loss function is defined as

$$Loss = \left(r_t + \gamma \max Q\left(s_{t+1}, a'; \omega'\right) - Q(s_t, a_t; \omega)\right)^2, \quad (9)$$

where γ is the discount factor. $Q(s_t, a_t; \omega)$ is the output of "EvalNet" that calculates the current Q-value of the state-action pair, and ω is the weight of DNNs of "EvalNet". $\max Q(s_{t+1}, a'; \omega')$ is the output of "TargetNet" that calculates the maximum Q-value when executing the action a' at the next state s_{t+1}, and ω' is the weight of DNNs of "TargetNet".

The main steps of the proposed DQN-based prediction algorithm of offloading operations are presented in Algorithm 3. First, the relevant parameters of the proposed algorithm are initialized (Line 3). For each epoch, the current offloading plan DEP_t, the current state s_t, and the current response time T_t are initialized (Lines 5∼6). While all tasks have not been completed (Line 7), the valid action space of DEP_t is obtained by Algorithm 2 (Line 8), and the action a_t in the valid action space is selected by using the ϵ-greedy strategy (Line 9), which determines the tasks to be scheduled and their execution locations. Next, the action a is executed and the new response time T_{t+1} is obtained (Line 10). Accordingly, the reward r_t is calculated and the current response time T is updated (Line 11), and then the next state s_{t+1} is received (Line 12). Next, (s_t, a_t, r_t, s_{t+1}) is stored into the replay memory (Line 13), where m samples are drawn randomly and Q_target_j is calculated (Line 14). These samples may come from different runtime environments, which ensures the adequacy of learning. During the calculation process of $\max Q(s_{j+1}, a'; \omega')$, Algorithm 2 is also used to obtain the valid action space in DEP_{j+1} and select the action with the maximum Q-value in this space. Then, the weight ω of "EvalNet" is updated by using the Adam optimizer according to the loss function (Line 15), and the weight ω' of "TargetNet" is updated every C iterations (Line 16). Finally, the current state is updated (Line 17).

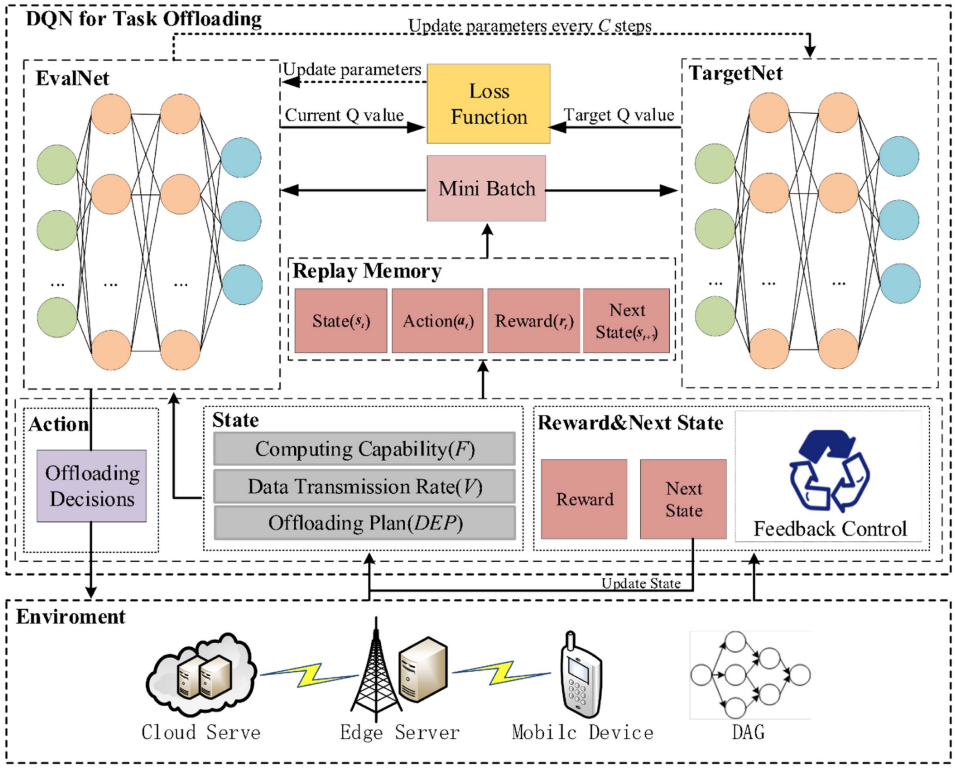


Fig. 2. Framework of DODQ for task offloading in the cloud-edge environment.

B. Runtime Decisions for Offloading Operations

The process of decision-making for offloading operations is conducted at runtime, where the main steps are presented in Algorithm 4. For each task of a mobile application, the current offloading plan DEP_t is first initialized (Line 6). Next, Algorithm 2 is used to obtain the valid action space A' of DEP_t (Line 9). Next, by using the Q-value prediction model, the Q-values of each action that meets constraints are evaluated and stored into the Q-value list Q_list (Lines 10∼13). Finally, the scheduling operation with the maximum Q-value in Q_list is selected and the offloading plan is updated (Lines 14∼16). Therefore, during the decision-making process, the execution locations of each task are determined in turn.

By the iterative feedback-control process, the optimal offloading plan can be gradually conducted in the runtime cloud-edge environment. The feedback control will continue until the scheduling plan of all tasks of an application are determined. Finally, the tasks of the application will be executed via a parallel manner according to the scheduling plan SCH_n output by Algorithm 4.

V. Performance Evaluation

In this section, we first introduce the simulation settings. Next, the proposed DODQ is evaluated with the consideration of the following three research questions (RQs).

- *RQ 1:* How much is the performance improvement of the DODQ compared to the state-of-art methods? (Section V-B)
- *RQ 2:* How much performance gap between the DODQ and the ideal plans? (Section V-C)
- *RQ 3:* How is the convergence performance of the DODQ? (Section V-D)

For *RQ 1*, the DODQ outperforms the PPO-based, PSO-GA, and Q-learning methods averagely by 6.89%, 3.48%, and 17.01% under different task scales, respectively. Specially, the DODQ presents much better performance than other methods with the increasing task scales. Meanwhile, the DODQ can well adapt to dynamic cloud-edge environments and generate the offloading plan in milliseconds. For *RQ 2*, the performance of the DODQ is comparable with the ideal plans under different scenarios, where their performance gap is less than 2%. Meanwhile, the DODQ can select offloading operations with the correctness of 92% under the support of the DQN-based Q-value prediction model. For *RQ 3*, the DODQ can achieve quick convergence and keep stable.

A. Simulation Settings

To simulate the diversity of applications, four different structures of DAGs are constructed with various task scales according to [38], [39], [40], where $n = \{10, 15, 20, 25\}$, as illustrated in Fig. 3. It is noted that some tasks in the DAGs can only be executed on mobile devices (highlighted in gray) but cannot be offloaded to the edge/cloud server.

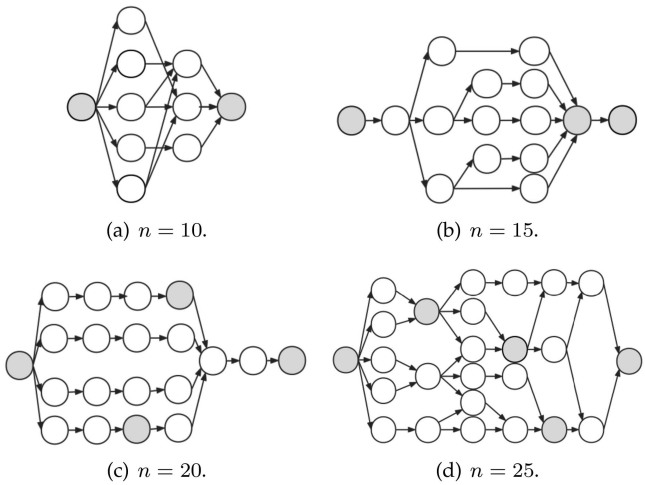
In each DAG, the calculation amount of the task i ($i \in N$) ranges from 50 to 500 Mcycles. For each dataflow edge $e_{i,j}$ ($e_{i,j} \in E$), the size of data transmission from the task i to

Algorithm 3: DQN-Based Prediction of Offloading Operations in the Cloud-Edge Environment.

Input: F, V, and $G(N, E)$.
Output: The Q-value prediction model.
Initialize The replay memory with the capacity M, the discount factor γ, the learning rate of the Adam optimizer α, the weight ω of "EvalNet", and the weight $\omega' = \omega$ of "TargetNet".
for each epoch **do**
  Initialize $DEP_t = (0, 0, \ldots, 0)$ and the current latency $T_t = 0$;
  Generate the current state $s_t = (F, V, DEP_t)$;
  while not all tasks have been scheduled **do**
    Calculate the valid action space A' of DEP_t based on Algorithm 2;
    Select an action $a_t \in A'$ by using the ϵ-greedy;
    Execute a_t and obtain the new latency T_{t+1} based on Algorithm 1;
    Calculate the reward $r_t = T_t - T_{t+1}$ and update the current latency $T_t = T_{t+1}$;
    Receive the next state $s_{t+1} = P(s_t, a_t)$;
    Store (s_t, a_t, r_t, s_{t+1}) into the replay memory;
    Draw m samples $\{(s_j, a_j, r_j, s_{j+1}) | 1 \leq j \leq m\}$ from the replay memory and calculate Q_target: get the $DEP_(j+1)$ from $s_(j+1)$; Calculate the valid action space A'' of DEP_{j+1} based on Algorithm 2;
    $Q\_target_j = \begin{cases} r_j, \text{ all tasks have been scheduled in } s_{j+1} \\ r_j + \gamma \max Q(s_{j+1}, a'; \omega')(a' \in A''), \text{ others} \end{cases}$;
    Update the weight ω of "EvalNet" by using the Adam optimizer according to the loss function: $Loss = \frac{1}{m} \sum_{j=1}^{m} (Q\_target_j - Q(s_j, a_j; \omega))^2$;
    Update the weight $\omega' = \omega$ of "TargetNet" every C iterations;
    Update the current state $s_t = s_{t+1}$;
  end
end

Algorithm 4: Runtime Decision-Making Process for Offloading Operations.

Input: F, V, and $G(N, E)$.
Output: SCH_n.
Declaration
  $Q\_value[a]$: the Q-value of a.
  Q_list: the Q-value list that meets constraints.
  $getQvalue()$: call the Q-value prediction model.
Initialize: $DEP_t = DEP_0 = (0, 0, \ldots, 0)$.
while not all tasks have been scheduled **do**
  $Q\_list = [\,]$;
  Call Algorithm 2 $\rightarrow$ valid action space A' of DEP_t;
  for each $a \in A'$ **do**
    Evaluate the Q-value of a by calling the Q-value prediction model: $Q\_value[a] = getQvalue(F, V, DEP_t, a)$;
    $Q\_list.add(Q\_value[a])$;
  end
  Select the action with the maximum Q-value: $a = A.getAction\_MaxQvalue(Q\_list)$;
  $DEP_{t+1} = Schedule(DEP_t, a)$ and update SCH_t;
  $DEP_t = DEP_{t+1}$;
end



(a) $n = 10$. (b) $n = 15$. (c) $n = 20$. (d) $n = 25$.

Fig. 3. Different structures of DAGs with various task scales.

the task j ranges from 0 to 1000 KB. Moreover, the computing capabilities of different nodes $F = (f_{MD}, f_{ES}, f_{CS})$ and the data transmission rates between these nodes $V = (v_{MD,ES}, v_{MD,CS}, v_{ES,CS})$ are subject to the corresponding uniform distributions. The detailed settings of simulation parameters are listed in Table IV.

The proposed DODQ is implemented based on TensorFlow 2.3.0 [41]. In DODQ, the fully-connected DNNs are used that consist of one input layer, two hidden layers, and one output layer, where these two hidden layers both have 128 hidden neurons. The memory capacity M, the training batch size m, the discount factor γ, and the learning rate of the Adam optimizer are set to 15000, 64, 0.9, and 0.001, respectively. Moreover, different training epochs are used for various DAGs, where the training epochs for the DAGs with different task scales (i.e., n=10, 15, 20, and 25) are set to 10000, 15000, 20000, and 25000, respectively.

Based on the above settings, five different scenarios of cloud-edge environments are simulated with various F and V, as shown in Table V. For each scenario, the runtime decision-making algorithm is executed to conduct adaptive task offloading in different cloud-edge environments.

B. RQ 1: Performance Improvement of the DODQ Compared to the State-of-Art Methods

The performance of the proposed DODQ is compared with three state-of-the-art methods to further evaluate its advantage for task offloading, which are described as follows.

- The PPO-based method [17]. It adopts the PPO-based DRL algorithm to train offloading strategies. Different from

TABLE IV
SETTINGS OF SIMULATION PARAMETERS

Parameter	Value	Parameter	Value
n	$\{10, 15, 20, 25\}$	f_{CS}	$U[5.0, 7.0]$ GHz
c_i	$[50, 500]$ Mcycles	$v_{MD,ES}$	$U[4.0, 8.0]$ MB/s
$d_{i,j}$	$[0, 1000]$ KB	$v_{MD,CS}$	$U[1.5, 2.5]$ MB/s
f_{MD}	$U[1.5, 2.5]$ GHz	$v_{ES,CS}$	$U[2.5, 3.5]$ MB/s
f_{ES}	$U[3.5, 4.5]$ GHz		

TABLE V
DIFFERENT SCENARIOS OF CLOUD-EDGE ENVIRONMENTS

No.	F (GHz)			V (MB/s)		
	f_{MD}	f_{ES}	f_{CS}	$v_{MD,ES}$	$v_{MD,CS}$	$v_{ES,CS}$
1	2.4	4.5	6.4	7.0	2.5	3.1
2	2.2	4.3	6.2	6.5	2.3	3.4
3	2.0	4.0	6.0	6.0	2.0	3.0
4	1.8	3.8	5.4	5.5	1.8	2.6
5	1.6	3.6	5.6	5.0	1.6	2.8

the proposed DODQ, it prioritizes the tasks in the DAGs following the descending order of the rank value of each task and then determines the execution locations of each task in turn. The hyper-parameter settings are the same as the DODQ in Section V-A.

- The PSO-GA-based method [26]. It introduces the crossover and mutation operators of GA to improve the particle update strategy of the traditional PSO algorithm, where each particle contains the scheduling order and execution locations of tasks. The start values of the two acceleration coefficients are both set to 0.9. The end values of the two acceleration coefficients are set to 0.2 and 0.4, respectively. The maximum and minimum values of the inertia weight are set to 0.9 and 0.4, respectively. Moreover, the iteration number and population number are set to 1000 and 200, respectively.
- The Q-learning-based method [30]. As an RL algorithm, it stores each state-action pair and its corresponding Q-values into a Q-table for maximizing the accumulative rewards of an offloading plan. This method selects a task from the DAG and determines the scheduling location. If an action does not meet the task dependency, severe punishment will be given in the setting of the reward function. The learning rate, discount factor, greedy probability, and the maximum training epochs are set to 0.01, 0.9, 0.1, and 200000, respectively. The method will return the offloading plan when the result keeps constant for 50000 consecutive iterations. When the runtime environments change, the Q-learning needs to retrain the decision-making model of task offloading for better fitting in a new environment.

As shown in Fig. 4, the DODQ outperforms the PPO-based method in terms of the response time by around 1.02%, 4.24%, 8.10%, and 11.74% with different task scales (i.e., n =10, 15, 20, and 25) under various scenarios, respectively. This is because the PPO-based method presets the order of task scheduling by prioritizing tasks, which limits the solution space of getting a better scheduling plan. With the increase of task scales, the optional scheduling order grows non-linearly, and thus it is hard to obtain the optimal scheduling order through presetting the task priority. For example, when $n = 20$ and 25, the DODQ is obviously superior to the PPO-based method.

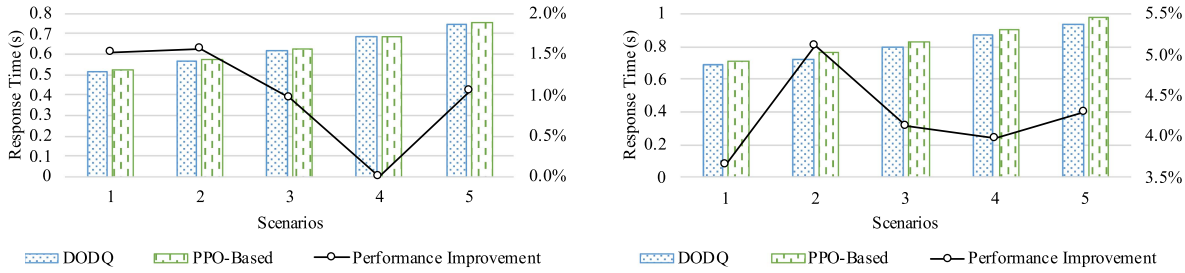
TABLE VI
DECISION-MAKING TIME (S) OF DIFFERENT METHODS WITH DIFFERENT TASK SCALES

n	DODQ	PPO-based	PSO-GA	Q-learning
10	0.29	0.31	40.18	21.42
15	0.39	0.42	127.89	33.49
20	0.51	0.54	233.49	54.75
25	0.61	0.63	329.74	92.76

As shown in Fig. 5, when $n = 10$ or 15, the DODQ and the PSO-GA achieve equivalent performance. When $n = 20$ and 25, the DODQ outperforms the PSO-GA by around 5.70% and 8.01%, respectively. This is because the quality of the solutions obtained by the PSO-GA might be seriously affected by the size of the solution space. If the solution space is small, the PSO-GA can obtain a high-quality plan through the cross and mutation operations. For example, when $n = 10$, the PSO-GA is able to search the optimal plan under various scenarios. However, when facing a large solution space, the PSO-GA is prone to fall into the local optimum and cannot achieve the optimal plan (e.g., when $n = 20$ and 25).

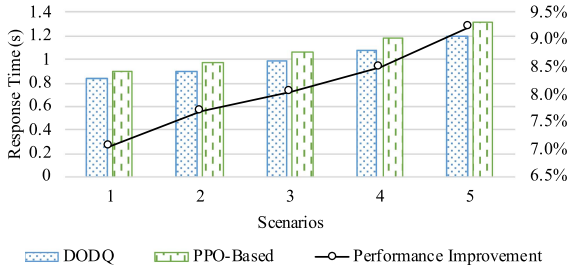
As shown in Fig. 6, the DODQ outperforms the Q-learning in terms of the response time by around 9.01%, 13.32%, 20.49%, and 25.23% with different task scales (i.e., $n = 10$, 15, 20, and 25) under various scenarios, respectively. This is because the Q-learning records all state-action pairs and their corresponding Q-values into a Q-table. When the task scale is small, the Q-learning can fully cover the state during the training process and obtain a fully-trained Q-table that contains the optimal plan. However, as the rising of task scales, it is hard for the Q-learning to achieve the optimal plan. Moreover, the Q-learning gives penalties in the reward function to avoid actions that do not meet the constraints, which caused massive mutation of Q-values in the learning process, and thus it is hard for the Q-learning to obtain good scheduling policies.

Furthermore, we evaluate the decision-making time of the DODQ, PPO-based, PSO-GA, and Q-learning methods with different task scales, where the results are listed in Table VI. Specifically, the decision-making time of the above methods under different task scales is 0.29∼0.61 s, 0.31∼0.63 s, 40.18∼329.74 s, and 21.42∼92.76 s, respectively. Similar to the DODQ, the PPO-based method implements millisecond-level runtime decisions. However, the PPO-based method needs to prioritize tasks before making decisions, and thus the decision-making time of the PPO-based method would be slightly longer than the DODQ. The decision-making time of the PSO-GA is seriously affected by the size of solution space. When facing a large solution space, the PSO-GA needs to spend massive time to search for a high-quality plan. Moreover, the Q-learning suffers from the problem of high-dimensional state space especially when the task scale becomes large because it records all state-action pairs and their corresponding Q-values into a Q-table. Meanwhile, when the

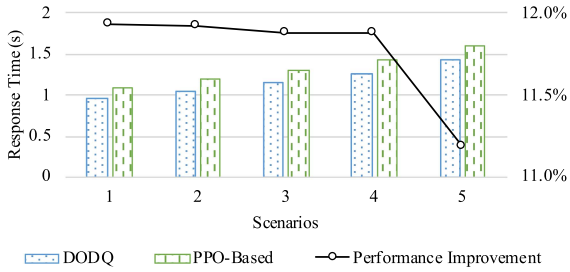


(a) $n = 10$.

(b) $n = 15$.

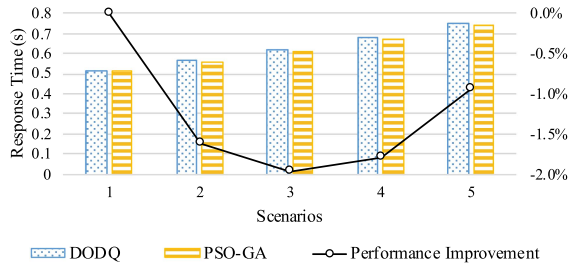


(c) $n = 20$.

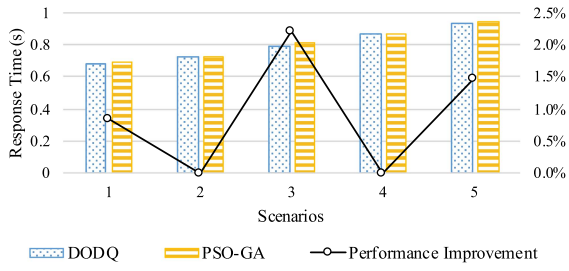


(d) $n = 25$.

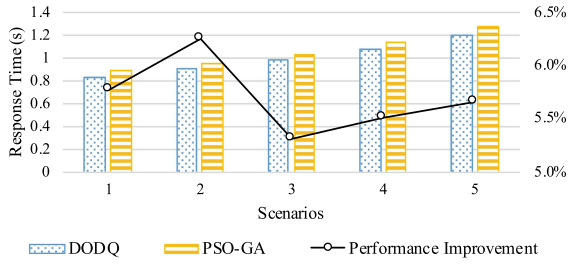
Fig. 4. Performance comparison between the DODQ and PPO-based method.



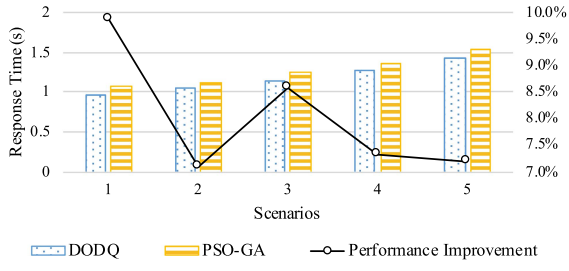
(a) $n = 10$.



(b) $n = 15$.



(c) $n = 20$.



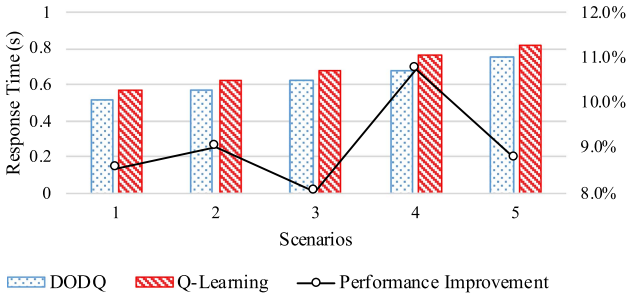
(d) $n = 25$.

Fig. 5. Performance comparison between the DODQ and PSO-GA method.

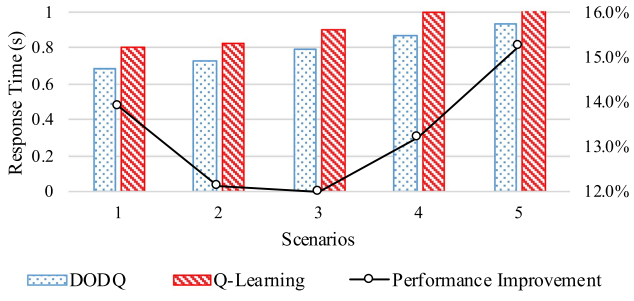
runtime environments change, the Q-learning needs to retrain the decision-making model of task offloading from scratch for better adaptability. These factors cause the excessive decision-making time of the Q-learning. The above results demonstrate the advantage of the DODQ in achieving low response time and high training efficiency.

C. RQ 2: Performance Gap Between the DODQ and the Ideal Plans

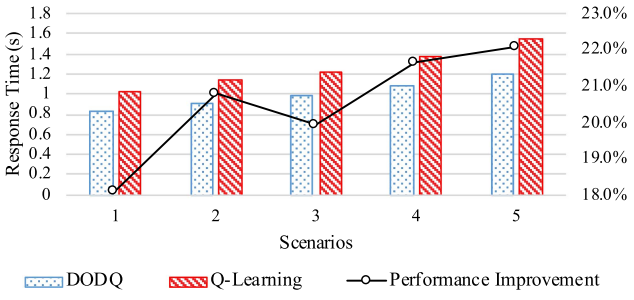
We test the performance gap between the DODQ and ideal plans, where the plans with the minimum response time are selected as the ideal ones by trying all possible plans. However, this may cause the problem of combination explosion when enumerating all possible plans. On one hand, the order of scheduling tasks in an application is not fixed and there may be different priorities when scheduling parallel tasks. On the other hand, after the priority of parallel tasks is given, each task might be executed on different locations (e.g., the mobile device and edge/cloud server). Therefore, the size of solution space depends on the number of task orders and execution locations, which is extremely huge. For example, when n =10 and 15, the optional task orders are 1680 and 124740, respectively; when n =20 and 25, the optional task orders have exceeded 60 million



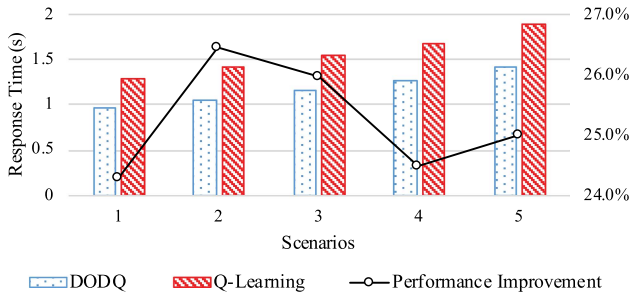
(a) $n = 10$.



(b) $n = 15$.



(c) $n = 20$.



(d) $n = 25$.

Fig. 6. Performance comparison between the DODQ and Q-learning method.

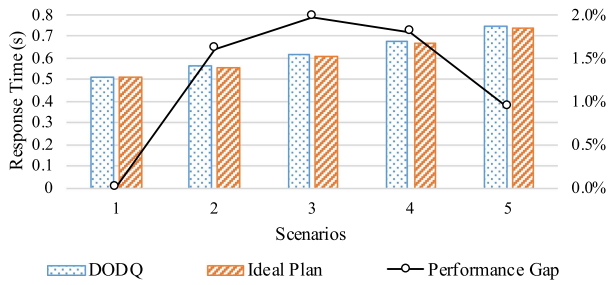


Fig. 7. Performance comparison with ideal plans when $n = 10$.

and 2 billion, respectively. Meanwhile, for each order of task scheduling, there are 3^{n-l} different execution locations, where l is the number of l_{task}. In the experiments, we can only get the ideal plan when $n = 10$ in the limited time, and it is noted that the solution space has reached $124740 * 3^{12}$ when $n = 15$. As shown in Fig. 7, the DODQ achieves comparable response time with the ideal plans under different scenarios, where the performance gap between the DODQ and the ideal plans always keeps less than 2%.

Next, The action accuracy rate (AAR) is used to measure the correctness of offloading operations during the decision-making process, which is defined as

$$AAR = \frac{A}{O}, \tag{10}$$

TABLE VII
AAR UNDER DIFFERENT SCENARIOS WHEN $n = 10$

Scenario	1	2	3	4	5
AAR	100%	90%	90%	90%	90%

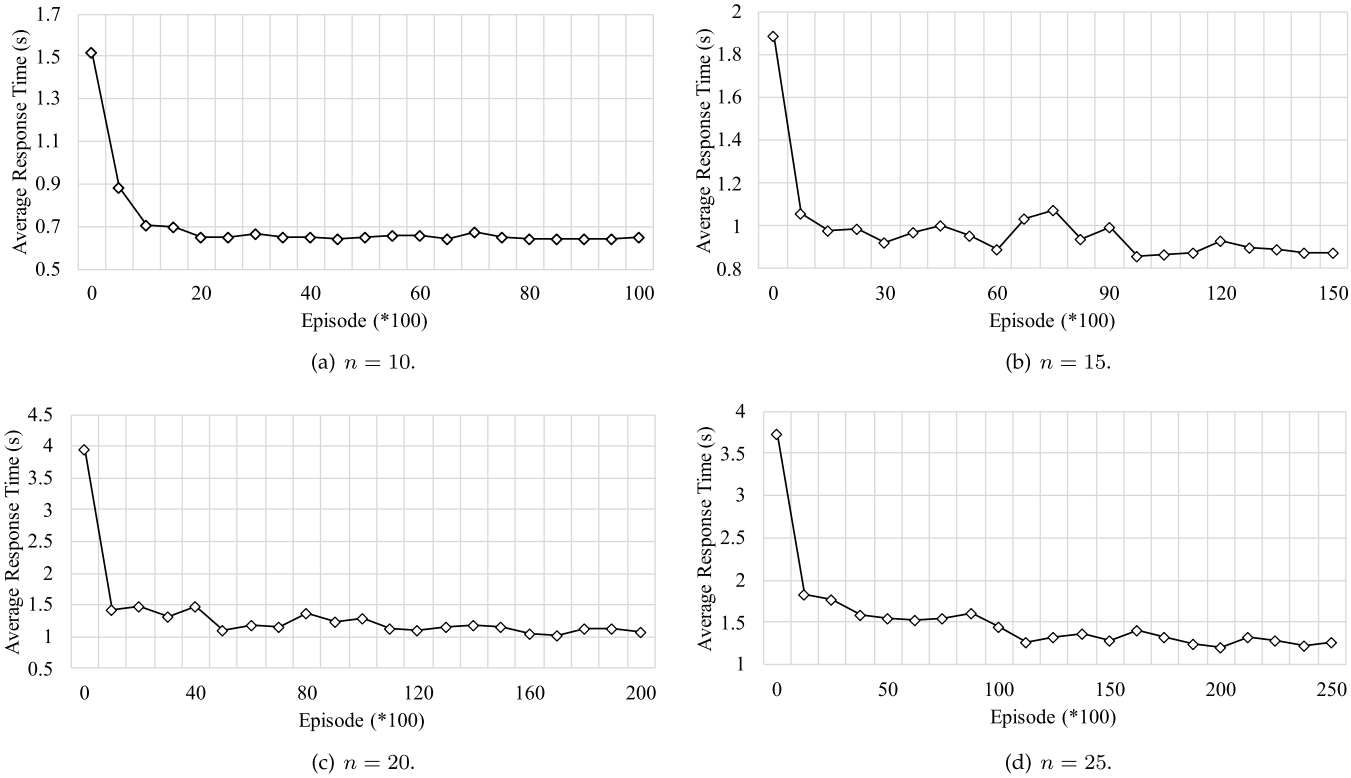
where O and A are the numbers of all executed offloading operations and the correct ones, respectively. When the offloading operations make the current offloading plan close to the ideal one, they are considered to be correct.

As shown in Table VII, the DODQ achieves the ideal solution in the Scenario 1, where all the decision-making of actions are correct. In the other four scenarios, the AAR of the DODQ reach 90% although the ideal plans are not obtained, where only one action has made an error. The above results verify the effectiveness of the DODQ in achieving optimal/near-optimal performance of task offloading in different cloud-edge environments.

Taking Scenario 1 ($n = 10$) as an example to explain the task offloading process by using the DODQ. As shown in Table VIII, after initializing $DEP_t = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$, the Q-values of the actions that meet constraints are stored into Q_list (as described in Algorithm 4), where $task_1^{MD}$ is the only action that meets constraints. Therefore, the task 1 is executed on the mobile device and then DEP_t transfers to (1, 0, 0, 0, 0, 0, 0, 0, 0, 0). After updating Q_list, the action with the highest Q-value (i.e., $task_3^{CS}$) is selected from Q_list and executed at each step. Therefore, the task 3 is offloaded to the cloud server

TABLE VIII
TASK OFFLOADING PROCESS IN SCENARIO 1 ($n = 10$)

DEP_t	**Actions with Top-3 Q-values in Q_list**		
0, 0, 0, 0, 0, 0, 0, 0, 0, 0	$task_1^{MD}(-0.0302)$	–	–
1, 0, 0, 0, 0, 0, 0, 0, 0, 0	$task_3^{CS}(-0.3387)$	$task_6^{MD}(-0.3389)$	$task_4^{CS}(-0.3392)$
1, 0, 3, 0 0, 0, 0, 0, 0, 0	$task_5^{MD}(-0.2359)$	$task_4^{ES}(-0.2380)$	$task_6^{MD}(-0.2459)$
1, 0, 3, 0, 1, 0, 0, 0, 0, 0	$task_4^{ES}(-0.2297)$	$task_6^{MD}(-0.2425)$	$task_7^{CS}(-0.2485)$
1, 0, 3, 2, 1, 0, 0, 0, 0, 0	$task_2^{ES}(-0.2290)$	$task_6^{MD}(-0.2320)$	$task_7^{CS}(-0.2411)$
1, 2, 3, 2, 1, 0, 0, 0, 0, 0	$task_6^{MD}(-0.1949)$	$task_7^{CS}(-0.1978)$	$task_9^{ES}(-0.2009)$
1, 2, 3, 2, 1, 1, 0, 0, 0, 0	$task_8^{MD}(-0.1887)$	$task_7^{CS}(-0.2034)$	$task_9^{ES}(-0.2055)$
1, 2, 3, 2, 1, 1, 0, 1, 0, 0	$task_9^{ES}(-0.0641)$	$task_7^{CS}(-0.0719)$	$task_9^{MD}(-0.1878)$
1, 2, 3, 2, 1, 1, 0, 1, 2, 0	$task_7^{CS}(-0.0592)$	$task_7^{ES}(-0.2526)$	$task_7^{MD}(-0.3837)$
1, 2, 3, 2, 1, 1, 3, 1, 2, 0	$task_{10}^{MD}(-0.0593)$	–	–
1, 2, 3, 2, 1, 1, 3, 1, 2, 1	–	–	–



(a) $n = 10$. (b) $n = 15$. (c) $n = 20$. (d) $n = 25$.

Fig. 8. Convergence performance of the DODQ.

for execution and then DEP_t transfers to (1, 0, 3, 0, 0, 0, 0, 0, 0, 0). Similarly, the action with the highest predicted Q-value in Q_list is selected and executed at each step. Finally, when $DEP_t = (1, 2, 3, 2, 1, 1, 3, 1, 2, 1)$, there is no more action to be executed and the decision-making process of task offloading is completed.

D. RQ 3: Convergence Performance of the DODQ

As shown in Fig. 8, we test the convergence performance of the DODQ in terms of the average response time under different scenarios. In general, the average response time of applications decreases as the number of training episodes increases. Specifically, it can be observed that the average response time declines significantly after 20% of the training episodes, which reflects that the DODQ can quickly learn suitable task offloading plans. Meanwhile, the DODQ converges after 80% of the training episodes and the average response time becomes stable.

VI. DISCUSSION

A. Problem Complexity

This paper studies the offloading problem of dependent and parallel tasks in cloud-edge environments, where the constraint relationships between tasks and the node heterogeneity make this problem extremely complex. Specifically, there are two key steps when dealing with this problem: the first step is to determine the sequence of task scheduling, and the second step

is to schedule tasks to appropriate nodes. For the first step, it is necessary to recursively search for all possible topological sorting of a DAG and select the optimal sequence from them, and the complexity of this step is $O((n+e)*n!)$ [42], where n and e are the numbers of vertices (i.e., the number of tasks) and edges in a DAG, respectively. For the second step, it is necessary to enumerate all possible task scheduling plans, and the complexity of this step is $O(k^n)$, where k is the number of nodes. Therefore, the problem complexity reaches $O((n+e)*n!*k^n)$, which should be an NP-hard problem. Existing methods cannot well handle this extremely-complex problem, because they require massive time to search for a suitable plan and need to restart searching when the environment changes.

B. Evaluation in Real-World Environments

We establish a cloud-edge testbed to simulate and evaluate the performance of the proposed DODQ in real-world environments, where the cloud and edge nodes are equipped with different computing capabilities and the data transmission rates between these nodes vary according to their locations. The results verify the effectiveness of the DODQ. It is noted that there are certain differences between the cloud-edge testbed and real-world environments. First, the applications run in a single-device environment, and thus the execution time of tasks on the same computing node approximates the average. Second, the mobility model of MDs is simplified and the wireless channel fading caused by the mobility is ignored, and thus the network conditions between an MD and the same computing node approximates the average. Despite the above factors, the DODQ can still work well in real-world environments just with some performance differences.

VII. Conclusion

In this paper, we propose the DODQ to explore the fast and adaptive task offloading in cloud-edge computing. First, we formulate the task offloading as an optimization problem of minimizing the response time of mobile applications that are modeled as DAGs. Next, the DQN is customized to train the decision-making model of task offloading that considers the arbitrary order of task scheduling when scheduling parallel tasks, which can well adapt to different cloud-edge environments and efficiently generate offloading plans. Extensive simulation results verify the good adaptability of the proposed DODQ. Specifically, the DODQ outperforms the PPO-based, PSO-GA, and Q-learning methods averagely by 6.89%, 3.48%, and 17.01% under different task scales, respectively. Meanwhile, the DODQ can generate offloading plans in milliseconds when the environment changes. Moreover, the DODQ reaches the optimal/near-optimal performance, where the performance gap between the DODQ and the ideal plans is always less than 2% under different scenarios. Besides, the DODQ can determine offloading operations with the average correctness of 92% and achieve quick and stable convergence. In our future work, we will extend the proposed model and utilize game-theoretic methods to solve the dynamic and complex offloading problems in multi-device and mobility-aware cloud-edge scenarios.

References

[1] J. Wang, J. Liu, and N. Kato, "Networking and communications in autonomous driving: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1243–1274, Second Quarter 2019.

[2] M. Hassaballah and S. Aly, "Face recognition: Challenges, achievements and future directions," *IET Comput. Vis.*, vol. 9, no. 4, pp. 614–626, 2015.

[3] K. Kim, M. Billinghurst, G. Bruder, H. B. Duh, and G. F. Welch, "Revisiting trends in augmented reality research: A review of the 2nd decade of ISMAR (2008–2017)," *IEEE Trans. Visual. Comput. Graph.*, vol. 24, no. 11, pp. 2947–2962, Nov. 2018.

[4] J. Huang, Y. Zhou, Z. Ning, and H. Gharavi, "Wireless power transfer and energy harvesting: Current status and future prospects," *IEEE Wireless Commun.*, vol. 26, no. 4, pp. 163–169, Aug. 2019.

[5] P. Cong, J. Zhou, L. Li, K. Cao, T. Wei, and K. Li, "A survey of hierarchical energy optimization for mobile edge computing: A perspective from end devices to the cloud," *ACM Comput. Surv.*, vol. 53, no. 2, pp. 1–44, 2020.

[6] A. Shakarami, M. Ghobaei-Arani, M. Masdari, and M. Hosseinzadeh, "A survey on the computation offloading approaches in mobile edge/cloud computing environment: A stochastic-based perspective," *J. Grid Comput.*, vol. 18, no. 4, pp. 639–671, 2020.

[7] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, Third Quarter 2017.

[8] H. Wu, W. J. Knottenbelt, and K. Wolter, "An efficient application partitioning algorithm in mobile environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 7, pp. 1464–1480, Jul. 2019.

[9] S. Deng, L. Huang, J. Taheri, and A. Y. Zomaya, "Computation offloading for service workflow in mobile cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 12, pp. 3317–3329, Dec. 2015.

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

[11] J. Yan, S. Bi, Y. J. Zhang, and M. Tao, "Optimal task offloading and resource allocation in mobile-edge computing with inter-user task dependency," *IEEE Trans. Wireless Commun.*, vol. 19, no. 1, pp. 235–250, Jan. 2020.

[12] J. Zhu, X. Li, R. Ruiz, and X. Xu, "Scheduling stochastic multi-stage jobs to elastic hybrid cloud resources," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 6, pp. 1401–1415, Jun. 2018.

[13] Z. Chen et al., "Multiobjective cloud workflow scheduling: A multiple populations ant colony system approach," *IEEE Trans. Cybern.*, vol. 49, no. 8, pp. 2912–2926, Aug. 2019.

[14] P. Kaur and S. Mehta, "Resource provisioning and work flow scheduling in clouds using augmented shuffled frog leaping algorithm," *J. Parallel Distrib. Comput.*, vol. 101, pp. 41–50, 2017.

[15] Y.-H. Jia et al., "An intelligent cloud workflow scheduling system with time estimation and adaptive ant colony optimization," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 51, no. 1, pp. 634–649, Jan. 2021.

[16] Y. Xie et al., "A novel directional and non-local-convergent particle swarm optimization based workflow scheduling in cloud-edge environment," *Future Gener. Comput. Syst.*, vol. 97, pp. 361–378, 2019.

[17] J. Wang, J. Hu, G. Min, W. Zhan, Q. Ni, and N. Georgalas, "Computation offloading in multi-access edge computing using a deep sequential model based on reinforcement learning," *IEEE Commun. Mag.*, vol. 57, no. 5, pp. 64–69, May 2019.

[18] J. Yan, S. Bi, and Y. J. A. Zhang, "Offloading and resource allocation with general task graph in mobile edge computing: A deep reinforcement learning approach," *IEEE Trans. Wireless Commun.*, vol. 19, no. 8, pp. 5404–5419, Aug. 2020.

[19] A. Zhu et al., "Computation offloading for workflow in mobile edge computing based on deep q-learning," in *Proc. Wireless Opt. Commun. Conf.*, 2019, pp. 1–5.

[20] Z. Peng, J. Lin, D. Cui, Q. Li, and J. He, "A multi-objective trade-off framework for cloud resource scheduling based on the deep q-network algorithm," *Cluster Comput.*, vol. 23, no. 4, pp. 2753–2767, 2020.

[21] F. Song, H. Xing, X. Wang, S. Luo, P. Dai, and K. Li, "Offloading dependent tasks in multi-access edge computing: A multi-objective reinforcement learning approach," *Future Gener. Comput. Syst.*, vol. 128, pp. 333–348, 2022.

[22] A. Yousafzai, A. Gani, R. M. Noor, A. Naveed, R. W. Ahmad, and V. Chang, "Computational offloading mechanism for native and android runtime based mobile applications," *J. Syst. Softw.*, vol. 121, pp. 28–39, 2016.

[23] Z. Cheng, P. Li, J. Wang, and S. Guo, "Just-in-time code offloading for wearable computing," *IEEE Trans. Emerg. Topics Comput.*, vol. 3, no. 1, pp. 74–83, Mar. 2015.
[24] Y. Wang and X. Zuo, "An effective cloud workflow scheduling approach combining PSO and idle time slot-aware rules," *IEEE/CAA J. Automatica Sinica*, vol. 8, no. 5, pp. 1079–1094, May 2021.
[25] Z. Tong, H. Chen, X. Deng, K. Li, and K. Li, "A novel task scheduling scheme in a cloud computing environment using hybrid biogeography-based optimization," *Soft Comput.*, vol. 23, no. 21, pp. 11035–11054, 2019.
[26] W. Guo, B. Lin, G. Chen, Y. Chen, and F. Liang, "Cost-driven scheduling for deadline-based workflow across multiple clouds," *IEEE Trans. Netw. Serv. Manage.*, vol. 15, no. 4, pp. 1571–1585, Dec. 2018.
[27] D. Cui, W. Ke, Z. Peng, and J. Zuo, "Multiple dags workflow scheduling algorithm based on reinforcement learning in cloud computing," in *Proc. Int. Symp. Comput. Intell. Intell. Syst.*, Springer, 2015, pp. 305–311.
[28] W. Jiahao, P. Zhiping, C. Delong, L. Qirui, and H. Jieguang, "A multi-object optimization cloud workflow scheduling algorithm based on reinforcement learning," in *Proc. Int. Conf. Intell. Comput.*, Springer, 2018, pp. 550–559.
[29] A. Nascimento, V. Olimpio, V. Silva, A. Paes, and D. de Oliveira, "A reinforcement learning scheduling strategy for parallel cloud-based workflows," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops*, 2019, pp. 817–824.
[30] S. Pan, Z. Zhang, Z. Zhang, and D. Zeng, "Dependency-aware computation offloading in mobile edge computing: A reinforcement learning approach," *IEEE Access*, vol. 7, pp. 134742–134753, 2019.
[31] L. Qian, Y. Wu, F. Jiang, N. Yu, W. Lu, and B. Lin, "NOMA assisted multi-task multi-access mobile edge computing via deep reinforcement learning for industrial Internet of Things," *IEEE Trans. Ind. Informat.*, vol. 17, no. 8, pp. 5688–5698, Aug. 2021.
[32] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4005–4018, Jun. 2019.
[33] M. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: Enabling interactive perception applications on mobile devices," in the *Proc. 9th Int. Conf. Mobile Syst., Appl., Serv.*, ACM, 2011, pp. 43–56.
[34] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, Fourth Quarter 2017.
[35] V. Mnih et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[36] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.
[37] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
[38] W. Zhang and Y. Wen, "Energy-efficient task execution for application as a general topology in mobile cloud computing," *IEEE Trans. Cloud Comput.*, vol. 6, no. 3, pp. 708–719, Third Quarter 2018.
[39] G. Xie, Y. Chen, Y. Liu, Y. Wei, R. Li, and K. Li, "Resource consumption cost minimization of reliable parallel applications on heterogeneous embedded systems," *IEEE Trans. Ind. Informat.*, vol. 13, no. 4, pp. 1629–1640, Aug. 2017.
[40] Q. Wu, F. Ishikawa, Q. Zhu, Y. Xia, and J. Wen, "Deadline-constrained cost optimization approaches for workflow scheduling in clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 12, pp. 3401–3412, Dec. 2017.
[41] M. Abadi et al., "Tensorflow: A system for large-scale machine learning," in the *Proc. 12th Symp. Operating Syst. Des. Implementation*, USENIX, 2016, pp. 265–283.
[42] Y. Inoue and S. Minato, "An efficient method for indexing all topological orders of a directed graph," in *Proc. 25th Int. Symp. Algorithms Comput.*, Springer, 2014, pp. 103–114.

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

Shengxi Hu received the BS degree in measurement and control from Fujian Normal University, China, in 2020. He is currently working toward the MS degree in computer technology with the College of Computer and Data Science, Fuzhou University. Since September 2020, he has also been a part of the Fujian Key Laboratory of Network Computing and Intelligent Information Processing, Fuzhou University. His current research interests include cloud/edge computing and computation offloading.

Chujia Yu received the BS degree in engineering from the Xi'an University of Posts and Telecommunications, Xi'an, China, in 2022. She is currently working toward the MS degree in computer technology with the College of Computer and Big Data, Fuzhou University. Her current research interests include computation offloading, mobile edge computing, and cloud computing.

Zheyi Chen (Member, IEEE) received the MSc degree in computer science and technology from Tsinghua University, China, in 2017, and the PhD degree in computer science from the University of Exeter, U.K., in 2021. He is a professor and Qishan scholar with the College of Computer and Data Science, Fuzhou University, China. His research interests include cloud-edge computing, resource optimization, deep learning, and reinforcement learning. He has published more than 30 research papers in reputable international journals and conferences such as *IEEE Transactions on Parallel and Distributed Systems*, IEEE INFOCOM, *IEEE Transactions on Industrial Informatics*, *IEEE Communications Magazine*, *IEEE Transactions on Cloud Computing*, *IEEE IoT Journal*, and IEEE ICC.

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 with the Department of Computer Science within the Faculty of Environment, Science and Economy, 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.