Received May 31, 2021, accepted June 7, 2021, date of publication June 11, 2021, date of current version June 21, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3088124

Task Offloading and Resource Scheduling in Hybrid Edge-Cloud Networks

QI ZHANG, LIN GUI, (Member, IEEE), SHICHAO ZHU, AND XIUPU LANG
Department of Electronic Engineering, Shanghai Jiao Tong University, Shanghai 200240, China

Corresponding author: Lin Gui (guilin@sjtu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61671295, in part by the Shanghai Key Laboratory of Digital Media Processing and Transmission under Grant 6141B060303, and in part by the National Fundamental Research Key Project of China under Grant JCKY2017203B082.

ABSTRACT Computation-intensive mobile applications are explosively increasing and cause computation overload for smart mobile devices (SMDs). With the assistance of mobile edge computing and mobile cloud computing, SMDs can rent computation resources and offload the computation-intensive applications to edge clouds and remote clouds, which reduces the application completion delay and energy consumption of SMDs. In this paper, we consider the mobile applications with task call graphs and investigate the task offloading and resource scheduling problem in hybrid edge-cloud networks. Due to the interdependency of tasks, time-varying wireless channels, and stochastic available computation resources in the hybrid edge-cloud networks, it is challenging to make task offloading decisions and schedule computation frequencies to minimize the weighted sum of energy, time, and rent cost (ETRC). To address this issue, we propose two efficient algorithms under different conditions of system information. Specifically, with full system information, the task offloading and resource scheduling decisions are determined based on semidefinite relaxation and dual decomposition methods. With partial system information, we propose a deep reinforcement learning framework, where the future system information is inferred by long short-term memory networks. The discrete offloading decisions and continuous computation frequencies are learned by a modified deep deterministic policy gradient algorithm. Extensive simulations evaluate the convergence performance of ETRC with various system parameters. Simulation results also validate the superiority of the proposed task offloading and resource scheduling algorithms over baseline schemes.

INDEX TERMS Mobile edge computing, task offloading, resource scheduling, optimization algorithm, deep reinforcement learning.

I. INTRODUCTION

With the ever-increasing growth of smart mobile devices (SMDs), e.g., smart phones and Internet of Things devices, various computation-intensive applications are unprecedented development such as virtual reality, autonomous driving, and interactive gaming [1], [2]. Most of the applications are quite demanding in terms of real-time processing and high energy efficiency. However, SMDs have limited computation capabilities and battery capacities, and cannot effectively execute computation-intensive applications locally. Mobile cloud computing (MCC) has been considered as a prominent technology to alleviate the computation workloads of SMDs by offloading applications to remote clouds [3]. However, offloading applications from SMDs to remote clouds will incur long backhaul delay, which is usually unacceptable for latency-sensitive applications. Mobile edge computing (MEC) [4], [5] is presented to overcome the challenge by providing cloud computing capability in close proximity to SMDs. MEC provides low-latency computation service, whereas MCC has enough computation resources and can serve more SMDs. Hence, MCC and MEC are complementary and can assist SMDs in executing computation-intensive applications cooperatively [6], [7]. In this paper, the hierarchical MCC and MEC networks are referred to as hybrid edge-cloud networks. From the perspective of SMDs, offloading applications to edge clouds and remote clouds reduces the execution delay and energy consumption but

The associate editor coordinating the review of this manuscript and approving it for publication was Massimo Cafaro.

increases the monetary cost of using cloud servers [8], [9]. It is crucially important for SMDs to reduce energy, time, and monetary cost by efficient computation offloading and resource scheduling schemes.

There are two popular computation offloading models used in existing literature on MCC and MEC, which are referred to as binary offloading [10] and partial offloading [11], [12], respectively. For the binary offloading model, mobile applications are indivisible and have to be executed as a whole. As for the partial offloading model, the input data of mobile applications can be divided into several independent parts and executed at SMDs and cloud servers in parallel. Nevertheless, many mobile applications consist of multiple computation modules [13]–[15]. These computation modules with specific functions are referred to as tasks in this paper. For example, a video classification application from Facebook consists of multiple tasks including video track, audio track, count segments and so on [13]. In [14], the vehicular navigation application is decomposed into four tasks including controller, map, traffic and path. In Alibaba data trace, more than 75% of applications are involved with dependent tasks [15]. The tasks of an application that can be modeled by a task call graph are referred to as inter-dependent tasks in this paper. The dependency among tasks cannot be ignored, because before one task is executed, all its immediate predecessors must have already been completed. On the other hand, we can take advantage of the dependency relationships among tasks and design efficient task offloading and resource scheduling schemes to reduce the execution cost of applications.

However, there are still some challenges to address. Firstly, due to the dependency relationships among tasks, the offloading decision and computation resource scheduling of previous tasks will make influences on succeeding tasks, that is, the task offloading and resource scheduling decisions are coupled among tasks. Secondly, considering the time-varying property of wireless channels and available computation resources, the task offloading and resource scheduling is highly dynamic. The computation tasks of an application might be executed on multiple computation nodes. Furthermore, offloading tasks to edge clouds and remote clouds decreases the energy and time cost of SMDs but incurs extra monetary cost. To balance the cost of energy, time, and rent, the task offloading and resource scheduling decisions should be carefully determined.

In this paper, we investigate the task offloading and resource scheduling problem in hybrid edge-cloud networks. The dependency relationships among tasks and the time-varying property of wireless channels and available computation resources are taken into consideration. To minimize the weighted sum of energy, time, and rent cost, a mixed-integer nonlinear programming (MINLP) problem is formulated, which is generally NP-Hard. We propose two efficient algorithms to solve the MINLP problem under the conditions of full and partial system information. Specifically, when full system information is given, the wireless channel states and available computation resources at each task's execution time are known before making task offloading and resource scheduling decisions. In this case, we transform the original problem into an equivalent quadratic constrained quadratic programming (QCQP) and obtain a suboptimal solution by semidefinite relaxation (SDR) and dual decomposition methods. Under the condition of partial system information, only the system information of current task is given. Without full system information, we design a deep reinforcement learning (DRL) framework to learn the task offloading and resource scheduling decisions. The long short-term memory (LSTM) networks are further introduced into the DRL framework to predict the time-series system states.

A. MAIN CONTRIBUTIONS

In summary, the main contributions of this paper are listed as follows.

- We investigate the task offloading and resource scheduling problem for inter-dependent tasks in hybrid edge-cloud networks. The energy consumption, completion delay, and computation resource rental are jointly considered. A weighted sum cost minimization problem is formulated, which is an NP-hard problem with discrete task offloading variables and continuous resource scheduling variables.
- We propose two efficient algorithms to solve the weighted sum cost minimization problem under different conditions of system information. With full system information, we present an approximate task offloading algorithm based on SDR technique and then derive the computation frequencies given offloading decisions. With partial system information, we design a DRL framework with LSTM networks to predict the time-varying system information. The discrete offloading decisions and continuous computation frequencies are learned by a modified deep deterministic policy gradient algorithm (DDPG).
- We evaluate the convergence of proposed algorithms and the impacts of system parameters on the weight sum cost. Numerical results also validate the superiority of proposed algorithms compared to conventional task offloading and resource scheduling methods.

The remainder of this paper is organized as follows. We review the related works in Section II. The system model and problem formulation are presented in Section III. The task offloading and resource scheduling algorithm with full system information is proposed in Section IV. The task offloading and resource scheduling algorithm with partial system information is detailed in Section V. Simulation results are given in Section VI. Finally, we conclude this paper in Section VII.

II. RELATED WORKS

In recent years, computation offloading in hybrid edge-cloud networks have attracted significant attention from academia.

Wang *et al.* in [16] considered hierarchical mobile edge computing in IoT networks. To achieve the high computational throughput and low energy consumption, a large timescale workload prediction method was presented based on LSTM networks, while a small timescale optimization algorithm was proposed to allocate computation and communication resources. Ahn *et al.* in [17] introduced an edge-cloud interworking framework for video analysis applications. This framework aimed to minimize the monetary cost of cloud resource usage while meeting the deadlines for video analysis applications. In [18], the authors formulated a constrained multi-objective computation offloading model to minimize time and energy consumption simultaneously. Three evolutionary algorithms were designed to solve Pareto fronts based on the push and pull search framework. The authors in [19] studied the cost-aware computation offloading and resource allocation problem for latency limited services. A heuristic solution was proposed based on the single-user optimal solution. The previous works have studied the hierarchical computation offloading problems under various performance metrics, such as energy consumption and completion time. However, most of the existing works on hybrid edge-cloud networks neglected the differentiated rent cost of edge clouds and remote cloud. The total cost of energy consumption, completion time and resource rent has also not been investigated in these works.

Several recent studies have focused on the dependency relationships among tasks. Wang *et al.* in [20] developed a deep sequential model to address the challenges of task dependency and dynamic scenarios in MEC. A sequence-to-sequence neural network was utilized to infer the optimal task offloading policy. [3] and [21] investigated the energy-efficient computation offloading schemes for mobile cloud computing and multicore-based mobile devices, respectively. The authors in [3] optimized computation offloading selection, clock frequency control, and transmission power allocation by using the optimization decomposition approach. [21] presented a heuristic offloading decision and task scheduling algorithm including the initial scheduling phase and task reassignment phase. In [22], the authors studied the target helper selection and bandwidth allocation scheme for complex task-flows in UAVs-assisted edge computing. The on-policy and off-policy algorithms were designed to minimize the average mission response time based on multi-agent reinforcement learning. The authors in [23] investigated the dependent task offloading and service caching problem in mobile edge computing. For a special case with a homogeneous MEC, the proposed favorite successor based algorithm obtained an approximate solution with a constant competitive ratio. The existing literature has designed different task offloading and resource allocation algorithms for inter-dependent tasks. However, these works did not consider hierarchical network structures and the monetary cost of using edge servers and cloud servers. Thus, the mentioned algorithms cannot be applied to the hybrid edge-cloud networks.

Different from previous works, we consider the dependency relationships among tasks and differentiated rent prices of edge clouds and remote cloud in hybrid edge-cloud networks. To decrease SMDs' costs of completing applications, the weighted sum of energy consumption, completion time, and resource rent is presented as the performance metric. Considering the novel performance metric, dependency relationships among tasks, we propose two efficient task offloading and resource scheduling schemes for hybrid edge-cloud networks with full and partial system information.

III. SYSTEM MODEL AND PROBLEM FORMULATION

A. SYSTEM OVERVIEW

As shown in Fig. 1, we consider a hybrid edge-cloud network with multiple SMDs, edge clouds, and a remote cloud. SMDs connect to edge clouds through wireless links between access points (APs) and SMDs. The uploaded data received by APs can also be forwarded to the remote cloud via the Internet. The average network throughput between APs and remote cloud is denoted as R_w. We assume each SMD associates with one edge cloud and walks randomly in the coverage of this edge cloud during application execution.

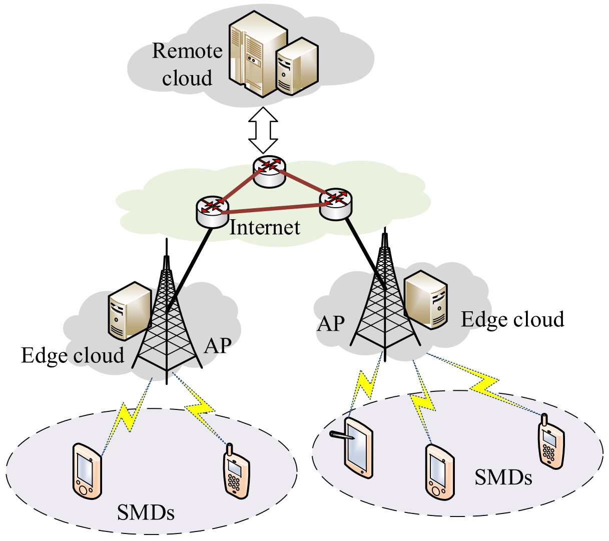


FIGURE 1. The architecture of a hybrid edge-cloud network with five SMDs, two edge clouds, and a remote cloud.

Each SMD has a computation-intensive application that can be partitioned into K inter-dependent tasks. Fig.2 gives an example of a video processing application with multiple inter-dependent tasks [13]. The dependencies of tasks can be represented by a task call graph $G = (\mathcal{K}, \mathcal{E})$, where $\mathcal{K}$ is the vertex set i.e., task set, and $\mathcal{E}$ is the edge set among tasks. Some previous works have studied the algorithms for generating task call graphs. For example, the authors in [13] considered the balance of parallelism and simplicity and proposed a dynamic algorithm to generate task call graphs for video processing applications. The task call graphs are designed by professional software designers of application service providers according to the functionalities of

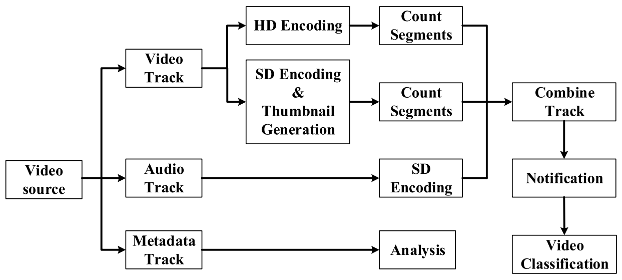


FIGURE 2. The task call graph of a video processing application.

applications and precedence constraints of tasks, instead of by SMDs or clouds. When applications are installed on SMDs, task call graphs are also stored in SMDs. Task call graphs can be modeled by adjacency matrixes that only take up a few storage resources of SMDs. As assumed in previous works [20]–[23], task call graphs are known by SMDs in this paper and we focus on the task offloading and resource scheduling scheme for inter-dependent tasks. We denote the workload of task k as w_k. The size of data transmitted from task j to task k is denoted as $d_{j,k}$. Each task can be executed at local, edge cloud or remote cloud, which are denoted by $\mathcal{N} = \{l, a, r\}$. Denote $x_k^l, x_k^a, x_k^r \in \{0, 1\}$ as the offloading decision variables of task k. $x_k^l = 1$, $x_k^a = 1$ and $x_k^r = 1$ indicate task k is executed at local, edge cloud, and remote cloud, respectively. To guarantee one task is executed only once, we have $x_k^l + x_k^a + x_k^r = 1$.

To simplify the formulation, we introduce two virtual tasks 0 and $K+1$ for applications as the entry and exit tasks. Task 0 is the immediate predecessor of task 1 and task $K + 1$ is the immediate successor of task K. The workloads of task 0 and $K + 1$ are zeros. The task set is denoted by $\mathcal{K} = \{0, 1, 2, \ldots, K, K+1\}$. Because the applications are initiated and terminated at SMDs, task 0 and task $K + 1$ need to be executed locally, i.e., $x_0^l = 1, x_{K+1}^l = 1$.

B. ENERGY CONSUMPTION MODEL FOR SMDs

The energy consumption of SMDs consists of the energy consumed on local execution and task offloading. Based on the dynamic voltage and frequency scaling technique [24], SMDs can adjust CPU-cycle frequencies to decrease the energy consumption of local execution. Denote the CPU-cycle frequency assigned to task k as f_k. The energy consumption of local execution can be calculated by $E_k^l = \kappa w_k (f_k)^2$, where κ is the effective capacitance coefficient depending on the chip architecture [12]. Then the total energy consumption of local execution can be written as:

$$E_l = \sum_{k\in\mathcal{K}} x_k^l \kappa \omega_k \left(f_k\right)^2 . \tag{1}$$

When task k is offloaded to edge clouds or remote cloud, frequency division multiple access (FDMA) is used in the uplink transmission from SMDs to APs, which fully mitigates the uplink interference [12]. Assume task j is an immediate predecessor of task k and it is executed locally. According to the Shannon formula, the uplink data rate from task j to task k is

$$R_{j,k}^u = W_u \log_2 \left(1 + \frac{P_l h_{j,k}^u}{n_0 W_u}\right), \tag{2}$$

where W_u is the uplink bandwidth, P_l is the transmit power of SMDs, $h_{j,k}^u$ is the time-varying uplink channel gain, n_0 is the power spectral density of additive Gaussian noise. The energy consumption of uplink transmission can be written as

$$E_u = \sum_{k\in\mathcal{K}} \sum_{j\in\mathbf{pred}(k)} \frac{x_j^l (1 - x_k^l) P_l d_{j,k}}{R_{j,k}^u}, \tag{3}$$

where $\mathbf{pred}(k)$ denotes the set of immediate predecessors of task k. The total energy consumption of one SMD when executing an application is $E = E_l + E_u$, i.e.,

$$E = \sum_{k\in\mathcal{K}} x_k^l \kappa \omega_k \left(f_k\right)^2 + \sum_{k\in\mathcal{K}} \sum_{j\in\mathbf{pred}(k)} \frac{x_j^l (1 - x_k^l) P_l d_{j,k}}{R_{j,k}^u}.$$

C. DELAY MODEL FOR COMPLETING APPLICATIONS

Assume that task j is executed at an edge cloud or remote cloud, task k is an immediate successor of task j and it is executed locally. Similar to the uplink transmission scheme, the downlink data rate from task j to task k can be formulated as

$$R_{j,k}^d = W_d \log_2 \left(1 + \frac{P_a h_{j,k}^d}{n_0 W_d}\right), \tag{4}$$

where W_d is the downlink bandwidth, P_a is the transmit power of APs, $h_{j,k}^d$ is the channel power gain from APs to SMDs. Due to the dependencies among tasks, one task cannot be executed until all its immediate predecessors have already been completed. We denote a path of an application as a sequence of tasks and edges from the entry task to the exit task. For example, $\{0, 1, 2, 8, 9\}$ is a path of an application as shown in Fig.6(a). To derive the completion delay of an application, we first define the completion delay of a path.

Definition 1: The completion delay of path i is defined as the sum of execution delay and transmission delay from entry task to exit task. Suppose the immediate predecessor of task k in path i is task j, task j and task k are executed at $m \in \mathcal{N}$ and $n \in \mathcal{N}$. Then, the completion delay of path i is given by

$$T_i = \sum_{k\in\mathbf{p}_i} \frac{w_k}{f_k} + \sum_{m\in\mathcal{N}} \sum_{n\in\mathcal{N}} \sum_{k\in\mathbf{p}_i\setminus\{0\}} x_j^m x_k^n d_{j,k} T_{j,k}, \tag{5}$$

where $\mathbf{p}_i$ is the task set in path i. $T_{j,k}$ is the transmission delay per bit of data from j to k, which is calculated as

$$T_{j,k} = \begin{cases} \frac{1}{R^u_{j,k}}, & if\, x^l_j = 1, x^l_k = 0 \\ \frac{1}{R^d_{j,k}}, & if\, x^l_j = 0, x^l_k = 1 \\ \frac{1}{R_w}, & if\, x^a_j x^r_k + x^r_j x^a_k = 1 \\ \frac{1}{R^u_{j,k}} + \frac{1}{R_w}, & if\, x^l_j = 1, x^r_k = 1 \\ \frac{1}{R^d_{j,k}} + \frac{1}{R_w}, & if\, x^r_j = 1, x^l_k = 1 \\ 0, & otherwise. \end{cases} \tag{6}$$

Since an application is completed when all tasks are executed, the completion delay of an application is equal to the maximal completion delay of all paths, i.e.,

$$T = \max_{i \in \mathcal{I}} \{T_i\}, \tag{7}$$

where $\mathcal{I}$ is the path set of an application.

D. PRICE MODEL FOR TASK OFFLOADING

When tasks are offloaded to edge clouds or remote cloud, SMDs need to rent computation resources from clouds. The edge clouds and remote cloud provide computation resources to SMDs with different prices. We consider a linear price function to represent the cost of renting computation resources [25]. Denote the unit price of computation resources at edge clouds and remote cloud as η^a and η^r, respectively. The total payment cost of each SMD about renting computation resources is given by

$$C = \sum_{k \in \mathcal{K}} (x^a_k \eta^a + x^r_k \eta^r) f_k. \tag{8}$$

E. PROBLEM FORMULATION

In this paper, we consider the energy-time-rent cost (ETRC) as the performance metric for each SMD, which is defined as the weighted sum of the energy consumption, completion time, and rent cost. We aim to determine where tasks should be executed and how much computation resources should be scheduled such that the ETRC of each SMD is minimized. Based on the above system model, the ETRC minimization problem for each SMD is formulated as a constrained optimization problem.

$$\begin{aligned} \mathcal{P}_1: \ \min_{\mathbf{x},\mathbf{f}} \quad & \varphi^e E + \varphi^t T + \varphi^c C & \text{(9a)} \\ \text{s.t.} \quad & T \le T_{\max} & \text{(9b)} \\ & 0 \le x^n_k f_k \le F^n_k, \quad \forall n \in \mathcal{N}, \quad \forall k \in \mathcal{K} & \text{(9c)} \\ & \sum_{n \in \mathcal{N}} x^n_k = 1, \quad \forall k \in \mathcal{K} & \text{(9d)} \\ & x^l_0 x^l_{K+1} = 1, & \text{(9e)} \\ & x^n_k \in \{0, 1\}, \quad \forall n \in \mathcal{N}, \quad \forall k \in \mathcal{K}, & \text{(9f)} \end{aligned}$$

where $\mathbf{x} = \{\mathbf{x}_k | k \in \mathcal{K}\} = \{x^n_k | k \in \mathcal{K}, n \in \mathcal{N}\}$, $\mathbf{f} = \{f_k | k \in \mathcal{K}\}$. $\varphi^e, \varphi^t, \varphi^c$ are weights of energy consumption, completion time, and rent cost, respectively. Constraint (9b) ensures the completion time of an application is bounded by the deadline $T_{\max}$. Constraint (9c) guarantees the computation resource scheduled for each task does not exceed available CPU cycle frequencies. Constraint (9d) implies that each task should be executed at only one computation node. Constraint (9e) ensures that virtual task 0 and $K + 1$ are executed at local. Constraint (9f) indicates x^n_k are binary variables.

IV. TASK OFFLOADING AND RESOURCE SCHEDULING WITH FULL SYSTEM INFORMATION

We first study the task offloading and resource scheduling algorithm with full system information, i.e., the wireless channel conditions and available computation resources of all tasks are given. The ETRC minimization problem $\mathcal{P}_1$ is a mixed-integer nonlinear programming problem, which is NP-hard. In order to reduce the computational complexity, we first transform $\mathcal{P}_1$ into an equivalent QCQP problem. Then, through the SDR approach, the QCQP problem can be converted into a standard convex problem which can be solved using convex optimization toolbox CVX [26]. Finally, we recover the binary offloading decisions by Gaussian randomization procedure and derive the solution of computation resource scheduling.

A. EQUIVALENT TRANSFORMATION INTO A QCQP PROBLEM

The complete time T is a maximum function of multiple paths, which might be a nonsmooth function. In order to transform the original nonsmooth objective function to a smooth one, we first introduce a new slack variable t and move the path delay T_i from objective function to the constraint: $T_i \le t, \forall i \in \mathcal{I}$. Next, we replace the integer constraint (9f) with quadratic constraints

$$x^n_k (x^n_k - 1) = 0, \quad \forall k \in \mathcal{K}, \quad n \in \mathcal{N}.$$

Let $g_k = (f_k)^2$, $u_k = \frac{1}{f_k}$, $T^u_{j,k} = \frac{1}{R^u_{j,k}}$, $T^d_{j,k} = \frac{1}{R^d_{j,k}}$, $T_w = \frac{1}{R_w}$, the original problem $\mathcal{P}_1$ can be equivalent to the following problem:

$$\begin{aligned} \mathcal{P}_2: \ \min_{\mathbf{x},\mathbf{f},t} \quad & \varphi^e \sum_{k \in \mathcal{K}} x^l_k \kappa w_k g_k \\ & + \varphi^e \sum_{k \in \mathcal{K}} \sum_{j \in \mathbf{pred}(k)} x^l_j (1 - x^l_k) P_l d_{j,k} T^u_{j,k} \\ & + \varphi^t t + \varphi^c \sum_{k \in \mathcal{K}} \left(x^a_k \eta^a + x^r_k \eta^r \right) f_k & \text{(10a)} \\ \text{s.t.} \quad & \sum_{k \in \mathbf{p}_i} w_k u_k + \sum_{m \in \mathcal{N}} \sum_{n \in \mathcal{N}} \sum_{k \in \mathbf{p}_i \backslash \{0\}} x^m_j x^n_k d_{j,k} T_{j,k} \\ & \le t, \quad \forall i \in \mathcal{I} & \text{(10b)} \\ & \sum_{k \in \mathbf{p}_i} w_k u_k + \sum_{m \in \mathcal{N}} \sum_{n \in \mathcal{N}} \sum_{k \in \mathbf{p}_i \backslash \{0\}} x^m_j x^n_k d_{j,k} T_{j,k} \end{aligned}$$

$$\le T_{\max}, \quad \forall i \in \mathcal{I} \tag{10c}$$
$$(f_k)^2 - g_k = 0, \quad \forall k \in \mathcal{K} \tag{10d}$$
$$f_k u_k = 1, \quad \forall k \in \mathcal{K} \tag{10e}$$
$$x_k^n(x_k^n - 1) = 0, \quad \forall n \in \mathcal{N}, \quad \forall k \in \mathcal{K}, \tag{10f}$$
$$(9c), (9d), (9e). \tag{10g}$$

Then, we transform problem $\mathcal{P}_2$ into a standard QCQP problem. First, we define auxiliary vectors $\mathbf{y}_k$ and vectorize the variables and parameters in $\mathcal{P}_2$ as follows.

$$\mathbf{y}_k = \left[x_k^l, x_k^a, x_k^r, f_k, g_k, u_k\right], \tag{11}$$
$$\mathbf{y} = [\mathbf{y}_0, \mathbf{y}_1 \cdots, \mathbf{y}_{K+1}, t]^T. \tag{12}$$

The objective function in $\mathcal{P}_2$ can be rewritten as

$$\mathbf{y}^T\mathbf{A}^e\mathbf{y} + \mathbf{y}^T\mathbf{B}^e\mathbf{y} + \mathbf{y}^T\mathbf{A}^c\mathbf{y} + (\mathbf{b}^t)^T\mathbf{y}, \tag{13}$$

where

$$\mathbf{A}^e = \begin{bmatrix} \mathbf{A}^e_{0,0} & & & & \\ & \mathbf{A}^e_{1,1} & & & \\ & & \ddots & & \\ & & & \mathbf{A}^e_{K+1,K+1} & \\ & & & & 0 \end{bmatrix},$$

$$\mathbf{A}^e_{k,k} = \frac{1}{2}\begin{bmatrix} \mathbf{0}_{4\times4} & \mathbf{a}^e_k & \mathbf{0}_{4\times1} \\ (\mathbf{a}^e_k)^T & 0 & 0 \\ \mathbf{0}_{1\times4} & 0 & 0 \end{bmatrix},$$

$$\mathbf{a}^e_k = \left[\varphi^e\kappa w_k, \mathbf{0}_{1\times3}\right]^T,$$

$$\mathbf{B}^e = \begin{bmatrix} \mathbf{0} & & & & \\ & \mathbf{0} & \cdots & \mathbf{B}^e_{j,k} & \\ & \vdots & \ddots & \vdots & \\ & \mathbf{B}^e_{k,j} & \cdots & \mathbf{0} & \\ & & & & 0 \end{bmatrix},$$

$$\mathbf{B}^e_{j,k} = \frac{1}{2}\varphi^e P_l d_{j,k} T^u_{j,k}\begin{bmatrix} 0 & 1 & 1 & \mathbf{0}_{1\times3} \\ & \mathbf{0}_{5\times6} & & \end{bmatrix},$$

$$\mathbf{B}^e_{k,j} = \left(\mathbf{B}^e_{j,k}\right)^T,$$

$$\mathbf{A}^c = \begin{bmatrix} \mathbf{A}^c_{0,0} & & & & \\ & \mathbf{A}^c_{1,1} & & & \\ & & \ddots & & \\ & & & \mathbf{A}^c_{K+1,K+1} & \\ & & & & 0 \end{bmatrix},$$

$$\mathbf{A}^c_{k,k} = \frac{1}{2}\begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{a}^c_k & \mathbf{0}_{3\times2} \\ (\mathbf{a}^c_k)^T & & \\ \mathbf{0}_{2\times3} & & \mathbf{0}_{3\times3} \end{bmatrix},$$

$$\mathbf{a}^c_k = \left[0, \varphi^c\eta^a, \varphi^c\eta^r\right]^T, \quad \mathbf{b}^t = \left[\mathbf{0}_{1\times6(K+2)}, \varphi^t\right]^T.$$

The constrain (10b) in $\mathcal{P}_2$ can be rewritten as

$$\mathbf{y}^T\mathbf{A}^{\mathbf{p}_i}\mathbf{y} + \mathbf{y}^T\mathbf{B}^{\mathbf{p}_i}\mathbf{y} - [\mathbf{e}_{6(K+2)+1}]^T\mathbf{y} \le 0, \quad \forall i \in \mathcal{I} \tag{14}$$

where

$$\mathbf{A}^{\mathbf{p}_i} = \begin{bmatrix} \mathbf{A}^{\mathbf{p}_i}_{0,0} & & & & \\ & \mathbf{A}^{\mathbf{p}_i}_{1,1} & & & \\ & & \ddots & & \\ & & & \mathbf{A}^{\mathbf{p}_i}_{K+1,K+1} & \\ & & & & 0 \end{bmatrix},$$

$$\mathbf{A}^{\mathbf{p}_i}_{k,k} = \begin{cases} \frac{1}{2}\begin{bmatrix} \mathbf{0}_{5\times5} & \mathbf{a}^{\mathbf{p}_i}_k \\ (\mathbf{a}^{\mathbf{p}_i}_k)^T & 0 \end{bmatrix}, & \text{if } k \in \mathbf{p}_i \\ \mathbf{0}_{6\times6}, & \text{otherwise.} \end{cases},$$

$$\mathbf{a}^{\mathbf{p}_i}_k = [w_k, w_k, w_k, \mathbf{0}_{1\times3}]^T,$$

$$\mathbf{B}^{\mathbf{p}_i} = \begin{bmatrix} \mathbf{0} & & & & \\ & \mathbf{0} & \cdots & \mathbf{B}^{\mathbf{p}_i}_{j,k} & \\ & \vdots & \ddots & \vdots & \\ & \mathbf{B}^{\mathbf{p}_i}_{k,j} & \cdots & \mathbf{0} & \\ & & & & 0 \end{bmatrix},$$

$$\mathbf{B}^{\mathbf{p}_i}_{k,j} = \left(\mathbf{B}^{\mathbf{p}_i}_{j,k}\right)^T,$$

$\mathbf{e}_k$ is a $[6(K+2)+1] \times 1$ standard unit vector with the k-th element being 1.

The constrain (10c) in $\mathcal{P}_2$ can be rewritten as

$$\mathbf{y}^T\mathbf{A}^{\mathbf{p}_i}\mathbf{y} + \mathbf{y}^T\mathbf{B}^{\mathbf{p}_i}\mathbf{y} - T_{\max} \le 0, \quad \forall i \in \mathcal{I}. \tag{16}$$

The constrain (10d) in $\mathcal{P}_2$ can be rewritten as

$$\mathbf{y}^T diag(\mathbf{e}_{6k+4})\mathbf{y} - (\mathbf{e}_{6k+5})^T\mathbf{y} = 0, \quad \forall k \in \mathcal{K}. \tag{17}$$

The constrain (10e) in $\mathcal{P}_2$ can be rewritten as

$$\mathbf{y}^T\mathbf{A}^u_k\mathbf{y} = 1, \quad \forall k \in \mathcal{K}, \tag{18}$$

$$\mathbf{B}^{\mathbf{p}_i}_{j,k} = \begin{cases} \frac{d_{j,k}}{2}\begin{bmatrix} 0 & \frac{1}{R^u_{j,k}} & \frac{1}{R^u_{j,k}} + \frac{1}{R_w} & \mathbf{0}_{1\times3} \\ \frac{1}{R^d_{j,k}} & 0 & \frac{1}{R_w} & \mathbf{0}_{1\times3} \\ \frac{1}{R^d_{j,k}} + \frac{1}{R_w} & \frac{1}{R_w} & 0 & \mathbf{0}_{1\times3} \\ \mathbf{0}_{3\times1} & \mathbf{0}_{3\times1} & \mathbf{0}_{3\times1} & \mathbf{0}_{3\times3} \end{bmatrix}, & \text{if } k \in \mathbf{p}_i \\ \mathbf{0}_{6\times6}, & \text{otherwise.} \end{cases} \tag{15}$$

where

$$\mathbf{A}_k^u = \begin{bmatrix} \mathbf{0} & & & & \\ & \ddots & & & \\ & & \mathbf{A}_{k,k}^u & & \\ & & & \ddots & \\ & & & & \mathbf{0} \\ & & & & & 0 \end{bmatrix},$$

$$\mathbf{A}_{k,k}^u = \frac{1}{2}\begin{bmatrix} \mathbf{0}_{5\times 5} & \mathbf{a}_k^u \\ (\mathbf{a}_k^u)^T & 0 \end{bmatrix}, \quad \mathbf{a}_k^u = [0,0,0,1,0]^T.$$

The constrain (10f) in $\mathcal{P}_2$ can be rewritten as

$$\mathbf{y}^T diag(\mathbf{e}_j)\mathbf{y} - (\mathbf{e}_j)^T\mathbf{y} = 0,$$
$$\forall j \in \{6k+1, 6k+2, 6k+3\}, \quad \forall k \in \mathcal{K}. \tag{19}$$

The constrain (9c) in $\mathcal{P}_2$ can be rewritten as

$$0 \le \mathbf{y}^T\mathbf{A}_k^n\mathbf{y} \le F_k^n, \quad \forall n \in \mathcal{N}, \quad \forall k \in \mathcal{K}, \tag{20}$$

where

$$\mathbf{A}_k^n = \begin{bmatrix} \mathbf{0} & & & & \\ & \ddots & & & \\ & & \mathbf{A}_{k,k}^n & & \\ & & & \ddots & \\ & & & & \mathbf{0} \\ & & & & & 0 \end{bmatrix},$$

$$\mathbf{A}_{k,k}^n = \frac{1}{2}\begin{bmatrix} \mathbf{0}_{3\times 3} & \mathbf{a}_k^n & \mathbf{0}_{3\times 2} \\ (\mathbf{a}_k^n)^T & & \\ \mathbf{0}_{2\times 3} & & \mathbf{0}_{3\times 3} \end{bmatrix},$$

$$\mathbf{a}_k^n = \begin{cases} [1,0,0]^T, & \text{if } n = l \\ [0,1,0]^T, & \text{if } n = a \\ [0,0,1]^T, & \text{otherwise.} \end{cases}$$

The constrain (9d) in $\mathcal{P}_2$ can be rewritten as

$$(\mathbf{b}_k^s)^T\mathbf{y} = 1, \quad \forall k \in \mathcal{K}, \tag{21}$$

where $\mathbf{b}_k^s = \mathbf{e}_{6k+1} + \mathbf{e}_{6k+2} + \mathbf{e}_{6k+3}, \forall k \in \mathcal{K}$.

The constrain (9e) in $\mathcal{P}_2$ can be rewritten as

$$\mathbf{y}^T\mathbf{A}^v\mathbf{y} = 1, \tag{22}$$

where $\mathbf{A}^v$ is a $[6(K+2)+1] \times [6(K+2)+1]$ matrix, the elements $[1, 6(K+1)+1]$ and $[6(K+1)+1, 1]$ are 0.5, and other elements are 0.

By further defining $\mathbf{z} = [\mathbf{y}^T \quad 1_{1\times 1}]^T$, $\mathcal{P}_2$ is transformed into the following equivalent QCQP formulation:

$$\begin{aligned}
\mathcal{P}_3: \min_{\mathbf{z}} \;& \mathbf{z}^T\mathbf{Q}^A\mathbf{z} && \text{(23a)}\\
\text{s.t.}\;& \mathbf{z}^T\mathbf{Q}^{\mathbf{p}_i}\mathbf{z} \le 0, \ \forall i \in \mathcal{I} && \text{(23b)}\\
& \mathbf{z}^T\mathbf{Q}^{t_i}\mathbf{z} \le T_{\max}, \ \forall i \in \mathcal{I} && \text{(23c)}\\
& \mathbf{z}^T\mathbf{Q}_k^g\mathbf{z} = 0, \ \forall k \in \mathcal{K} && \text{(23d)}\\
& \mathbf{z}^T\mathbf{Q}_k^u\mathbf{z} = 1, \ \forall k \in \mathcal{K} && \text{(23e)}\\
& \mathbf{z}^T\mathbf{Q}_j\mathbf{z} = 0, && \\
& \forall j \in \{6k+1, 6k+2, 6k+3\}, \quad \forall k \in \mathcal{K} && \text{(23f)}\\
& 0 \le \mathbf{z}^T\mathbf{Q}_k^n\mathbf{z} \le F_k^n, \quad \forall n \in \mathcal{N}, \quad \forall k \in \mathcal{K} && \text{(23g)}\\
& \mathbf{z}^T\mathbf{Q}_k^s\mathbf{z} = 1, \ \forall k \in \mathcal{K} && \text{(23h)}\\
& \mathbf{z}^T\mathbf{Q}^v\mathbf{z} = 1, && \text{(23i)}\\
& \mathbf{z} \succeq 0, && \text{(23j)}
\end{aligned}$$

where

$$\mathbf{Q}^A = \begin{bmatrix} \mathbf{A}^e + \mathbf{B}^e + \mathbf{A}^c & \frac{1}{2}\mathbf{b}^t \\ \frac{1}{2}(\mathbf{b}^t)^T & 0 \end{bmatrix},$$

$$\mathbf{Q}^{\mathbf{p}_i} = \begin{bmatrix} \mathbf{A}^{\mathbf{p}_i} + \mathbf{B}^{\mathbf{p}_i} & -\frac{1}{2}\mathbf{e}_{6(K+2)+1} \\ -\frac{1}{2}[\mathbf{e}_{6(K+2)+1}]^T & 0 \end{bmatrix},$$

$$\mathbf{Q}^{t_i} = \begin{bmatrix} \mathbf{A}^{\mathbf{p}_i} + \mathbf{B}^{\mathbf{p}_i} & \mathbf{0} \\ \mathbf{0} & 0 \end{bmatrix},$$

$$\mathbf{Q}_k^g = \begin{bmatrix} diag(\mathbf{e}_{6k+4}) & -\frac{1}{2}\mathbf{e}_{6k+5} \\ -\frac{1}{2}\mathbf{e}_{6k+5}^T & 0 \end{bmatrix}, \quad \mathbf{Q}_k^u = \begin{bmatrix} \mathbf{A}_k^u & \mathbf{0} \\ \mathbf{0} & 0 \end{bmatrix},$$

$$\mathbf{Q}_j = \begin{bmatrix} diag(\mathbf{e}_j) & -\frac{1}{2}\mathbf{e}_j \\ -\frac{1}{2}\mathbf{e}_j^T & 0 \end{bmatrix}, \quad \mathbf{Q}_k^n = \begin{bmatrix} \mathbf{A}_k^n & \mathbf{0} \\ \mathbf{0} & 0 \end{bmatrix},$$

$$\mathbf{Q}_k^s = \begin{bmatrix} \mathbf{0} & \frac{1}{2}\mathbf{b}_k^s \\ \frac{1}{2}(\mathbf{b}_k^s)^T & 0 \end{bmatrix}, \quad \mathbf{Q}^v = \begin{bmatrix} \mathbf{A}_v & \mathbf{0} \\ \mathbf{0} & 0 \end{bmatrix}.$$

However, the equivalent QCQP problem $\mathcal{P}_3$ is still non-convex and hard to solve. To make problem $\mathcal{P}_3$ tractable, we adopt the SDR technique and convert $\mathcal{P}_3$ into a standard convex problem.

B. SEMIDEFINITE RELAXATION

Define $\mathbf{Z} = [\mathbf{z}\mathbf{z}^T]_{(6(K+2)+2)\times(6(K+2)+2)}$ and then we have $\mathbf{z}^T\mathbf{Q}\mathbf{z} = \mathrm{Tr}(\mathbf{Q}\mathbf{Z})$ with $\mathrm{rank}(\mathbf{Z}) = 1$. The non-convex constraint $\mathrm{rank}(\mathbf{Z}) = 1$ is not considered temporarily and the problem $\mathcal{P}_3$ is relaxed into the following problem:

$$\begin{aligned}
\mathcal{P}_4: \min_{\mathbf{Z}} \;& \mathrm{Tr}(\mathbf{Q}^A\mathbf{Z}) && \text{(24a)}\\
\text{s.t.}\;& \mathrm{Tr}(\mathbf{Q}^{\mathbf{p}_i}\mathbf{Z}) \le 0, \ \forall i \in \mathcal{I} && \text{(24b)}\\
& \mathrm{Tr}(\mathbf{Q}^{t_i}\mathbf{Z}) \le T_{\max}, \ \forall i \in \mathcal{I} && \text{(24c)}\\
& \mathrm{Tr}(\mathbf{Q}_k^g\mathbf{Z}) = 0, \ \forall k \in \mathcal{K} && \text{(24d)}\\
& \mathrm{Tr}(\mathbf{Q}_k^u\mathbf{Z}) = 1, \ \forall k \in \mathcal{K} && \text{(24e)}\\
& \mathrm{Tr}(\mathbf{Q}_j\mathbf{Z}) = 0, && \\
& \forall j \in \{6k+1, 6k+2, 6k+3\}, \ \forall k \in \mathcal{K} && \text{(24f)}\\
& 0 \le \mathrm{Tr}(\mathbf{Q}_k^n\mathbf{Z}) \le F_k^n, \ \forall n \in \mathcal{N}, \ \forall k \in \mathcal{K} && \text{(24g)}\\
& \mathrm{Tr}(\mathbf{Q}_k^s\mathbf{Z}) = 1, \quad \forall k \in \mathcal{K} && \text{(24h)}\\
& \mathrm{Tr}(\mathbf{Q}_v\mathbf{Z}) = 1, && \text{(24i)}\\
& \mathbf{Z}[6(K+2)+2, 6(K+2)+2] = 1, && \text{(24j)}\\
& \mathbf{Z} \succeq 0. && \text{(24k)}
\end{aligned}$$

Now, we have transformed the original problem $\mathcal{P}_1$ to a convex optimization problem $\mathcal{P}_4$, which could be solved in polynomial time using standard CVX tools [26].

C. EXTRACTING OFFLOADING DECISIONS

Denote the optimal solution of $\mathcal{P}_4$ as $\mathbf{Z}^*$. Because $\mathbf{Z}^*$ is solved by dropping the rank constraint $\text{rank}(\mathbf{Z}) = 1$, we need to recover offloading decisions $\mathbf{x}$ of original problem $\mathcal{P}_1$ from $\mathbf{Z}^*$ such that the offloading decisions $\mathbf{x}$ are in the feasible region of $\mathcal{P}_1$. If $\mathbf{Z}^*$ is of rank one, the offloading decisions $\mathbf{x}$ are in the feasible region of $\mathcal{P}_1$ naturally and we have

$$\mathbf{Z}^* = \mathbf{z}^*\mathbf{z}^{*T} = \begin{bmatrix} \mathbf{y}^* \\ 1 \end{bmatrix} \begin{bmatrix} \mathbf{y}^{*T} & 1 \end{bmatrix}, \tag{25}$$

where $\mathbf{y}^* = \left[\mathbf{y}_0^*, \mathbf{y}_1^* \cdots, \mathbf{y}_{K+1}^*, t^*\right]^T$. According to the definition of $\mathbf{y}_k$ (11), we can extract the optimal offloading decision $\mathbf{x}_k^*$ for task k in the diagonal of $\mathbf{Z}^*$ where the subscripts are from $(6k+1)$ to $(6k+3)$. Computation frequencies f_k^* can be extracted in the diagonal of $\mathbf{Z}^*$ where the subscript is $(6k+4)$.

If the rank of $\mathbf{Z}^*$ is larger than one, we propose an algorithm based on Gaussian randomization [27] to obtain the approximate offloading decisions of $\mathcal{P}_1$. First, we extract 3×3 sub-matrixes $\{\mathbf{Z}_k | k \in [1, 2, \ldots, K]\}$ from $\mathbf{Z}^*$ where the elements' subscripts are from $(6k+1)$ to $(6k+3)$. Then, for any $k \in [1, 2, \ldots, K]$, we generate V random 3×1 vectors ξ_k^v from a multivariate Gaussian distribution with zero mean and covariance $\mathbf{Z}_k$, i.e., $\xi_k^v \sim \mathbf{N}(\mathbf{0}_{3\times 1}, \mathbf{Z}_k), \forall v \in \{1, \ldots, V\}$. In order to satisfy the constraint (9d) and (9f), we recover the offloading decision $\mathbf{x}_k^v$ by setting the largest element in ξ_k^v to be one, and the other elements are set to be zero.

D. COMPUTATION RESOURCE SCHEDULING ALGORITHM GIVEN TASK OFFLOADING DECISIONS

When the task offloading decisions $\mathbf{x}_k^v = \{x_k^l, x_k^a, x_k^r\}, \forall k \in \mathcal{K}$ are recovered, we next calculate the corresponding computation frequencies f_k^v. Let $\chi^e = \sum_{k\in\mathcal{K}} \sum_{j\in\mathbf{pred}(k)} \frac{x_j^l(1-x_k^l)P_l d_{j,k}}{R_{j,k}^u}$, $\chi_i^t = \sum_{m\in\mathcal{N}} \sum_{n\in\mathcal{N}} \sum_{k\in\mathbf{p}_i\setminus\{0\}} x_j^m x_k^n d_{j,k} T_{j,k}$, and $\chi_k^c = x_k^a \eta^a + x_k^r \eta^r$. Given $\mathbf{x}_k^v$, the χ^c, χ^e, and χ^t are all constants, then the problem $\mathcal{P}_2$ can be rewritten as follows.

$$\mathcal{P}_5: \min_{\mathbf{f},t} \quad \varphi^e \left[\sum_{k\in\mathcal{K}} x_k^l \kappa w_k (f_k^v)^2 + \chi^e\right] + \varphi^t t$$
$$+ \varphi^c \sum_{k\in\mathcal{K}} f_k^v \chi_k^c \tag{26a}$$
$$\text{s.t.} \quad \sum_{k\in\mathbf{p}_i} \frac{w_k}{f_k^v} + \chi_i^t \le t, \quad \forall i \in \mathcal{I} \tag{26b}$$
$$\sum_{k\in\mathbf{p}_i} \frac{w_k}{f_k^v} + \chi_i^t \le T_{\max}, \quad \forall i \in \mathcal{I} \tag{26c}$$
$$0 \le f_k^v \le F_k^n, \quad \forall k \in \mathcal{K}. \tag{26d}$$

It can be verified that $\mathcal{P}_5$ is a convex problem. However, due to coupling constraints (26b) and (26c), it is complex to obtain the optimal computation frequencies based on KKT conditions. Next, we resort to the dual decomposition method to solve $\mathcal{P}_5$. The partial Lagrangian of $\mathcal{P}_5$ can be written as

$$\begin{aligned}
L(\mathbf{f}, t, \lambda) &= \varphi^e \left[\sum_{k\in\mathcal{K}} x_k^l \kappa w_k (f_k^v)^2 + \chi^e\right] + \varphi^t t \\
&\quad + \varphi^c \sum_{k\in\mathcal{K}} f_k^v \chi_k^c + \sum_{i\in\mathcal{I}} \lambda_i \left(\sum_{k\in\mathbf{p}_i} \frac{w_k}{f_k^v} + \chi_i^t - t\right) \\
&\quad + \sum_{i\in\mathcal{I}} \mu_i \left(\sum_{k\in\mathbf{p}_i} \frac{w_k}{f_k^v} + \chi_i^t - T_{\max}\right) \\
&= \sum_{k\in\mathcal{K}} \left[x_k^l \varphi^e \kappa w_k (f_k^v)^2 + \varphi^c f_k^v \chi_k^c\right] \\
&\quad + \sum_{i\in\mathcal{I}} (\lambda_i + \mu_i) \sum_{k\in\mathbf{p}_i} \frac{w_k}{f_k^v} + \left(\varphi^t - \sum_{i\in\mathcal{I}} \lambda_i\right) t \\
&\quad - \sum_{i\in\mathcal{I}} \mu_i T_{\max} + \sum_{i\in\mathcal{I}} (\lambda_i + \mu_i) \chi_i^t + \varphi^e \chi^e \\
&= L(\mathbf{f}, \boldsymbol{\lambda}, \boldsymbol{\mu}) + L(t, \boldsymbol{\lambda}) - \sum_{i\in\mathcal{I}} \mu_i T_{\max} \\
&\quad + \sum_{i\in\mathcal{I}} (\lambda_i + \mu_i) \chi_i^t + \varphi^e \chi^e,
\end{aligned}$$

where $\boldsymbol{\lambda} = \{\lambda_i | i \in \mathcal{I}\}, \boldsymbol{\mu} = \{\mu_i | i \in \mathcal{I}\}$ are the Lagrangian multipliers. The dual function can be obtained as follows.

$$\begin{aligned}
g(\boldsymbol{\lambda}, \boldsymbol{\mu}) &= \inf_{\mathbf{f}} L(\mathbf{f}, \boldsymbol{\lambda}, \boldsymbol{\mu}) + \inf_t L(t, \boldsymbol{\lambda}) - \sum_{i\in\mathcal{I}} \mu_i T_{\max} \\
&\quad + \sum_{i\in\mathcal{I}} (\lambda_i + \mu_i) \chi_i^t + \varphi^e \chi^e.
\end{aligned} \tag{27}$$

As subproblem $\mathcal{P}_5$ is convex and Slater's condition holds, strong duality can be guaranteed. Hence we can obtain the optimal computation resource scheduling by solving the following dual problem.

$$\max_{\boldsymbol{\lambda}, \boldsymbol{\mu}} \quad g(\boldsymbol{\lambda}, \boldsymbol{\mu}), \quad \text{s.t.} \quad \lambda_i, \mu_i \ge 0, \quad \forall i \in \mathcal{I}. \tag{28}$$

Due to the convexity of $L(\mathbf{f}, \boldsymbol{\lambda}, \boldsymbol{\mu})$ and $L(t, \boldsymbol{\lambda})$, we can obtain $\inf_{\mathbf{f}} L(\mathbf{f}, \boldsymbol{\lambda}, \boldsymbol{\mu})$ and $\inf_t L(t, \boldsymbol{\lambda})$ by stationary point and boundary values F_k^n. Let the derivative of $L(\mathbf{f}, \boldsymbol{\lambda}, \boldsymbol{\mu})$, $L(t, \boldsymbol{\lambda})$ with respect to f_k^v, t be zero, we have

$$2\varphi^e x_k^l \kappa w_k f_k^v + \varphi^c \chi_k^c - \sum_{i\in\mathcal{I}(k)} \frac{(\lambda_i + \mu_i) w_k}{(f_k^v)^2} = 0,$$
$$\varphi^t - \sum_{i\in\mathcal{I}} \lambda_i = 0,$$

where $\mathcal{I}(k)$ is the path set which includes k-th task. Then we have

$$f_k^v = \begin{cases} \min\left\{ \sqrt[3]{\dfrac{\sum_{i\in\mathcal{I}(k)}(\lambda_i+\mu_i)}{2\varphi^e\kappa}}, F_k^l \right\}, & \text{if } x_k^l = 1 \\ \min\left\{ \sqrt{\dfrac{w_k\sum_{i\in\mathcal{I}(k)}(\lambda_i+\mu_i)}{\varphi^c\eta^a}}, F_k^a \right\}, & \text{if } x_k^a = 1 \\ \min\left\{ \sqrt{\dfrac{w_k\sum_{i\in\mathcal{I}(k)}(\lambda_i+\mu_i)}{\varphi^c\eta^r}}, F_k^r \right\}, & \text{if } x_k^r = 1 \end{cases} \tag{29}$$

$$\sum_{i\in\mathcal{I}} \lambda_i = \varphi^t. \tag{30}$$

To solve the master optimization problem (28) about $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$, we adopt the projected subgradient method to update the dual variables $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$. Firstly, for each $i \in \mathcal{I}$, λ_i and μ_i are updated by corresponding subgradients as follows.

$$\tilde{\lambda}_i^{\tau+1} = \lambda_i^\tau + \alpha_i^\tau \left(\sum_{k\in\mathbf{p}_i} \frac{w_k}{f_k^v} + \chi_i^t - t \right), \tag{31}$$

$$\mu_i^{\tau+1} = \left[\mu_i^\tau + \beta_i^\tau \left(\sum_{k\in\mathbf{p}_i} \frac{w_k}{f_k^v} + \chi_i^t - T_{\max} \right) \right]^+, \tag{32}$$

where $[x]^+ = \max\{x, 0\}$, α_i^τ and β_i^τ are the diminishing step sizes of λ_i and μ_i in τ-th iteration respectively and satisfy the square summable but not summable rule [28]. Secondly, to guarantee the feasibility of dual variables, the updated $\tilde{\lambda}_i^{\tau+1}$ should be projected into the feasible region given in (30). We calculate the dual variable $\lambda_i^{\tau+1}$ by minimizing the square of Euclidean distance between $\{\tilde{\lambda}_i^{\tau+1}, i \in \mathcal{I}\}$ and $\{\lambda_i^{\tau+1}, i \in \mathcal{I}\}$ in the feasible region as follows.

$$\mathcal{P}_6: \min_{\lambda_i^{\tau+1}} \sum_{i\in\mathcal{I}} \left(\lambda_i^{\tau+1} - \tilde{\lambda}_i^{\tau+1}\right)^2 \tag{33}$$

$$\text{s.t.} \sum_{i\in\mathcal{I}} \lambda_i^{\tau+1} = \varphi^t, \tag{34}$$

$$\lambda_i^{\tau+1} \geq 0, \quad \forall i \in \mathcal{I}. \tag{35}$$

This problem is convex and the optimal $\lambda_i^{\tau+1}$ can be obtained by Lagrange multiplier method. The partial Lagrangian of $\mathcal{P}_6$ can be written as

$$L(\lambda, \nu) = \sum_{i\in\mathcal{I}} \left(\lambda_i^{\tau+1} - \tilde{\lambda}_i^{\tau+1}\right)^2 + \nu \left(\sum_{i\in\mathcal{I}} \lambda_i^{\tau+1} - \varphi^t \right).$$

Let $\frac{\partial L(\lambda,\nu)}{\lambda_i^{\tau+1}} = 0$ and $\frac{\partial L(\lambda,\nu)}{\nu} = 0$, we can obtain

$$\lambda_i^{\tau+1} = \tilde{\lambda}_i^{\tau+1} - \frac{1}{|\mathcal{I}|} \left(\sum_{i\in\mathcal{I}} \tilde{\lambda}_i^{\tau+1} - \varphi^t \right), \tag{36}$$

where $|\mathcal{I}|$ is the number of all paths in an application. Due to $\lambda_i^{\tau+1} \geq 0, \forall i \in \mathcal{I}$, Hence, we have

$$\lambda_i^{\tau+1} = \left[\tilde{\lambda}_i^{\tau+1} - \frac{1}{|\mathcal{I}|} \left(\sum_{i\in\mathcal{I}} \tilde{\lambda}_i^{\tau+1} - \varphi^t \right) \right]^+. \tag{37}$$

Based on the above analysis, given the offloading decision $\mathbf{x}_k^v$, we can obtain the solution of computation resource scheduling by (29), (32), and (37) iteratively.

Finally, given the offloading decision $\{\mathbf{x}_k^v | k \in \mathcal{K}\}$, and corresponding computation frequency $\{f_k^v | k \in \mathcal{K}\}$, we can obtain the value of objective function (9a), denoted by U^v. By searching the minimum value of objective function U^v over all V randomizations, we can obtain the solution $\mathbf{x}, \mathbf{f}$ to $\mathcal{P}_1$. We summary the SDR-based task offloading and resource scheduling (STORS) algorithm in Alg.1.

V. TASK OFFLOADING AND RESOURCE SCHEDULING WITH PARTIAL SYSTEM INFORMATION

In Section III, we propose a task offloading and resource scheduling algorithm given full system information, including all states of the channel gain and available computation resource. However, in practice, it is not trivial to obtain the accurate full system information in advance. In this section, we introduce DRL technology to solve the time-dependent task offloading and resource scheduling problem. We aim to learn the offloading decisions $\mathbf{x}$ and computation frequencies $\mathbf{f}$ only with instantaneous system information of current task.

Due to the offloading decisions $\mathbf{x}$ are discrete and computation frequency decisions $\mathbf{f}$ are continuous, conventional reinforcement learning methods, such as Q-learning, DDPG, are difficult to learn the hybrid action space directly. In this section, we propose a DRL-based algorithm to solve this problem. First, we transform the task offloading and resource scheduling problem into a Markov decision process (MDP) and define the states, actions, and reward function of the MDP.

A. MDP MODELING

1) *States*: The system state s_k when executing task k is jointly determined by the dependencies, workload, data size of task k, the channel gains, and the available computation resources, which is defined as follows.

$$s_k = \{G_k, w_k, \mathbf{d}_{j,k}, \mathbf{h}_{jk}, F_k^l, F_k^a, F_k^r\} \in \mathcal{S},$$

where G_k is the k-th row of G, which represents the adjacent relationships of task k. $\mathbf{d}_{j,k} = \{d_{j,k} | j \in \mathbf{pred}(k)\}$, $\mathbf{h}_{j,k} = \{h_{j,k}^u, h_{j,k}^d | j \in \mathbf{pred}(k)\}$.

2) *Actions*: For each task k, $a_k = \{n_k, f_k\} \in \mathcal{A}$ consists of the offloading decision n_k and computation frequency f_k. $n_k \in \mathcal{N}$ indicates the computation node where task k is executed. By setting $x_k^{n_k} = 1$ and other elements in $\mathbf{x}_k$ are zeros, $\mathbf{x}_k$ can be derived from n_k. The action space is hybrid where n_k is the discrete action and f_k is the continuous action.

Algorithm 1 SDR-Based Task Offloading and Resource Scheduling Algorithm (STORS)

```
Input: $G, w_k, d_{j,k}, h^u_{j,k}, h^d_{j,k}, P_l, P_a, W_u, W_d, n_0, R_w, F^n_k, \eta^a, \eta^r, \varphi^e, \varphi^t, \varphi^c, \forall k \in \mathcal{K}, \forall j \in \textbf{pred}(k)$
Output: x, f
    Initialization: $\lambda^0_i = \frac{\varphi^t}{|\mathcal{I}|}, \mu^0_i = 0, \epsilon = 10^{-2}, \alpha^0_i = 1, \beta^0_i = 1$.
1:  Solve the optimal solution $\mathbf{Z}^*$ of SDP problem $\mathcal{P}_4$.
2:  if rank($\mathbf{Z}^*$) = 1 then
3:      for $k = 1$ to $K$ do
4:          Extract $\mathbf{x}^*_k$ in the diagonal of $\mathbf{Z}^*$ where the subscripts are from $(6k+1)$ to $(6k+3)$.
5:          Extract $f^*_k$ in the diagonal of $\mathbf{Z}^*$ where the subscript is $(6k+4)$.
6:      end for
7:      Obtain $\mathbf{x} = \{x^*_k|\forall k \in \mathcal{K}\}$, $\mathbf{f} = \{f^*_k|\forall k \in \mathcal{K}\}$.
8:  else
9:      Extract $3 \times 3$ sub-matrixs $\{\mathbf{Z}_k|k \in [1, 2, \ldots, K]\}$ from $\mathbf{Z}^*$ where the elements' subscripts are from $(6k+1)$ to $(6k+3)$.
10:     for $v = 1$ to $V$ do
11:         for $k$ =1 to $K$ do
12:             Generate $\xi^v_k \sim \mathbf{N}(\mathbf{0}_{3\times1}, \mathbf{Z}_k)$.
13:             Recover $\mathbf{x}^v_k$ by setting the largest element in $\xi^v_k$ to be one and the other elements to be zero.
14:         end for
15:         repeat
16:             Calculate $\{f^v_k|\forall k \in \mathcal{K}\}$ and $U^v_\tau$ via (29) and (9a).
17:             Update $\tilde{\lambda}^{\tau+1}_i$, $\mu^{\tau+1}_i$ and $\lambda^{\tau+1}_i$ by (31), (32) and (37).
18:             Update step sizes $\alpha^{\tau+1}_i = \frac{\alpha^0_i}{\tau}$, $\beta^{\tau+1}_i = \frac{\beta^0_i}{\tau}$.
19:         until $|U^v_\tau - U^v_{\tau-1}| < \epsilon$
20:         if $U^v_\tau < U^*_\tau$ then
21:             Update $U^*_\tau = U^v_\tau$, $\mathbf{x} = \{x^v_k|\forall k \in \mathcal{K}\}$, $\mathbf{f} = \{f^v_k|\forall k \in \mathcal{K}\}$.
22:         end if
23:     end for
24: end if
25: return x, f
```

3) *Reward*: The reward function maps the state space $\mathcal{S}$ and action space $\mathcal{A}$ to a real number, i.e., $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. To reflect the ETRC of task k in the objective function (9a), we define the reward obtained from completing task k as

$$\begin{aligned} r_k = &-\varphi^e x^l_k \kappa w_k (f_k)^2 \\ &- \varphi^e \sum_{j \in \textbf{pred}(k)} x^l_j (1 - x^l_k) P_l d_{j,k} T^u_{j,k} \\ &- \varphi^t \left(\frac{w_k}{f_k} + \max_{j \in \textbf{pred}(k)} \{d_{j,k} T_{j,k}\} \right) \\ &- \varphi^c \left(x^a_k \eta^a + x^r_k \eta^r \right) f_k - \zeta_k [T - T_{\max}]^+, \end{aligned} \tag{38}$$

where ζ_k is a penalty parameter. When completion delay T is larger than deadline $T_{\max}$, the reward r_k is reduced by $\zeta_k[T - T_{\max}]^+$, which makes the DRL-based algorithm meet the deadline constrain (9b).

B. DRL-BASED TASK OFFLOADING AND RESOURCE SCHEDULING ALGORITHM

In reinforcement learning, an agent's behavior is determined by a policy π, which is a stationary distribution of actions given states. The goal of one SMD is to learn a policy that maximizes the expected return from the entry task to exit task by interacting with the environment. The action-value function (also named Q function) is used in many reinforcement learning algorithms to evaluate the performance of a policy. Before defining the action-value function, we first give the definition of the return. The return G_k is the total discounted reward starting from task k,

$$G_k = \sum_{j=k}^{K+1} \gamma^{j-k} r_j, \tag{39}$$

where $\gamma \in [0, 1]$ is the discounted factor. The action-value function $Q_\pi(s_k, a_k)$ is defined as the expected return beginning from state s_k, taking action a_k, and following policy π thereafter, which is given by

$$Q_\pi(s_k, a_k) = \mathbb{E}_\pi(G_k|s_k, a_k). \tag{40}$$

1) ACTOR-CRITIC BASED HYBRID ACTION GENERATION

Because the action a_k consists of discrete part n_k and continuous part f_k, it is intractable to learn the hybrid action by existing methods [20], [24]. We divide the hybrid action space into two parts and propose a modified DDPG algorithm to determine the offloading decision n_k and computation frequency f_k. The modified DDPG algorithm is designed based on the *actor-critic* architecture [29]. For each task, the *actor* is used to generate continuous computation frequency f_k by a deterministic policy network $\mu(s_k|\theta^\mu)$, where θ^μ is the parameters of the policy network. The *critic* evaluates the output of the *actor* and also generates the discrete action n_k by an action-value network $Q(s_k, f_k, n_k|\theta^Q)$, where θ^Q is the parameters of the action-value network. The action-value network $Q(s_k, f_k, n_k|\theta^Q)$ takes state s_k and computation frequency f_k as inputs, and outputs three Q values related to computation nodes l, a, and r. Then, the computation node with the maximum Q values is selected as the optimal computation node, i.e.,

$$x^{\hat{n}_k}_k = \begin{cases} 1, & \text{if } \hat{n}_k = arg \max\limits_{n_k} Q(s_k, f_k, n_k|\theta^Q) \\ 0, & \text{otherwise.} \end{cases} \tag{41}$$

The maximum Q value related to computation node $\hat{n}_k$ represents the evaluation of the learned offloading decision and computation frequency. Hence, it is reasonable to regard the maximum Q value as the Q value of the policy μ, i.e.,

$$Q(s_k, f_k, \hat{n}_k|\theta^Q) = arg \max_{n_k} Q(s_k, f_k, n_k|\theta^Q). \tag{42}$$

2) ACTOR-CRITIC NETWORK UPDATING

Because the computation frequencies **f** obtained from the policy network μ are continuous, we update the policy network μ based on DDPG algorithm [30]. Define the expected return beginning from entry task as $J(\mu) = \mathbb{E}(G_0|\mu)$. According to the deterministic policy gradient theorem [31], the policy gradient $\nabla J_{\theta^\mu}(\mu)$ can be calculated by

$$\nabla J_{\theta^\mu}(\mu) = \mathbb{E}\left[\nabla_f Q(s_k, f_k, \hat{n}_k|\theta^Q)\nabla_{\theta^\mu}\mu(s_k|\theta^\mu)\right]. \quad (43)$$

To guarantee the samples are independently and identically distributed when updating neural networks, we use a replay buffer to store the transitions (s_k, a_k, r_k, s_{k+1}) and update the policy network μ by sampling mini-batch transitions from the replay buffer. The sampled policy gradient is given by:

$$\nabla J_{\theta^\mu}(\mu) \approx \frac{1}{M}\sum_{k\in\mathcal{M}}\left[\nabla_f Q(s_k, f_k, \hat{n}_k|\theta^Q)\nabla_{\theta^\mu}\mu(s_k|\theta^\mu)\right], \quad (44)$$

where M is the cardinality of $\mathcal{M}$, i.e., the number of sampled transitions.

Next, we elaborate on the process of updating the *critic* network. The optimal Q function is the maximum expected return, which can be reformulated by the Bellman equation:

$$\begin{aligned} Q(s_k, f_k, \hat{n}_k) \\ &= \mathbb{E}[r_k + \gamma \max_{n_{k+1}} Q(s_{k+1}, f_{k+1}, n_{k+1})|s_k, f_k, \hat{n}_k], \quad (45)\end{aligned}$$

where $s_{k+1}, f_{k+1}, n_{k+1}$ are the state, computation frequency, and offloading decision of task $k+1$, respectively. It implies the optimal *critic* policy should maximize the expected value of $r_k + \gamma \max_{n_{k+1}} Q(s_{k+1}, f_{k+1}, n_{k+1})$. Let $y_k = r_k + \gamma \max_{n_{k+1}} Q(s_{k+1}, f_{k+1}, n_{k+1}|\theta^Q)$ denote the target value in each iteration. The *critic* network can be updated by minimizing the following loss function:

$$L(\theta^Q) = \mathbb{E}[(y_k - Q(s_k, f_k, \hat{n}_k|\theta^Q))^2]. \quad (46)$$

Similar to the previous *actor* network, the *critic* network is also updated by sampling mini-batch transitions from the replay buffer. Then, by differentiating the loss function $L(\theta^Q)$ with respect to the parameters θ^Q, the gradient for updating the *critic* network is given by

$$\begin{aligned}\nabla_{\theta^Q} L(\theta^Q) \approx \frac{1}{M}\sum_{k\in\mathcal{M}}[(y_k - Q(s_k, f_k, n_k|\theta^Q)) \\ \nabla_{\theta^Q} Q(s_k, f_k, n_k|\theta^Q)]. \quad (47)\end{aligned}$$

The DRL-based learning framework is shown in Fig.3. As in DDPG [30], we use *Target actor* network and *Target critic* network to calculate the target Q value $Q(s_{k+1}, f_{k+1}, n_{k+1})$, which can stabilize the training process. The architectures of *Target actor* network and *Target critic* network are the same to *actor* network and *critic* network, respectively. The weights $\theta^{\mu'}$ and $\theta^{Q'}$ of target networks are modified by soft updates, i.e.,

$$\theta^{\mu'} \leftarrow \beta\theta^\mu + (1-\beta)\theta^{\mu'}, \quad (48)$$

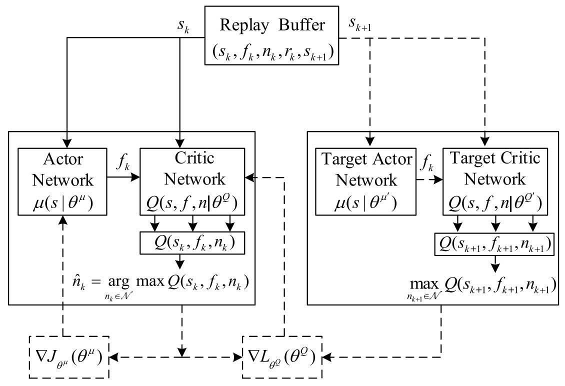

s_k
Replay Buffer
$(s_k, f_k, n_k, r_k, s_{k+1})$
s_{k+1}
Actor Network $\mu(s|\theta^\mu)$
f_k
Critic Network $Q(s,f,n|\theta^Q)$
$Q(s_k, f_k, n_k)$
$\hat{n}_k = \arg\max_{n_k\in\mathcal{N}} Q(s_k, f_k, n_k)$
Target Actor Network $\mu(s|\theta^{\mu'})$
Target Critic Network $Q(s,f,n|\theta^{Q'})$
$Q(s_{k+1}, f_{k+1}, n_{k+1})$
$\max_{n_{k+1}\in\mathcal{N}} Q(s_{k+1}, f_{k+1}, n_{k+1})$
$\nabla J_{\theta^\mu}(\theta^\mu)$
$\nabla L_{\theta^Q}(\theta^Q)$

FIGURE 3. The DRL-based learning framework.

$$\theta^{Q'} \leftarrow \beta\theta^Q + (1-\beta)\theta^{Q'}. \quad (49)$$

where $\beta \ll 1$. The solid lines in Fig.3 indicate the actor-critic based action generation and the dash lines show the process of actor-critic network updating.

3) NEURAL NETWORK ARCHITECTURE

Due to the temporal dependence, tasks are executed sequentially in time series. One of the widely used methods to learn the temporal dependence of sequential observations is the recurrent neural network (RNN) [32], [33]. Because it contains a self-recurrent loop that facilitates transporting information from the previous state to the next state. LSTM is a specially designed RNN model which contains the memory block for long-timescale prediction, we use the LSTM model [34] to learn the temporal states of hybrid edge-cloud networks.

Fig.4 illustrates the neural network architecture of the *actor* network. The Input layer is responsible for taking the state as input and passing them to the following layers. The LSTM layer is responsible for learning the dynamics of channel gains and available computation resources, and predicting them in the near future. The FC layer is responsible

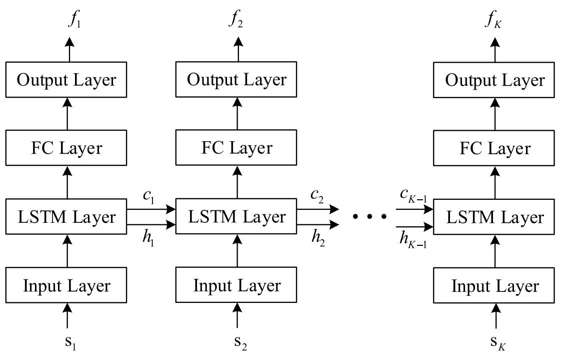


FIGURE 4. The architecture of *actor* network.

for learning the *actor* policy and mapping the outputs of LSTM layers to computation frequencies. The Output layer is responsible for output the learned computation frequencies.

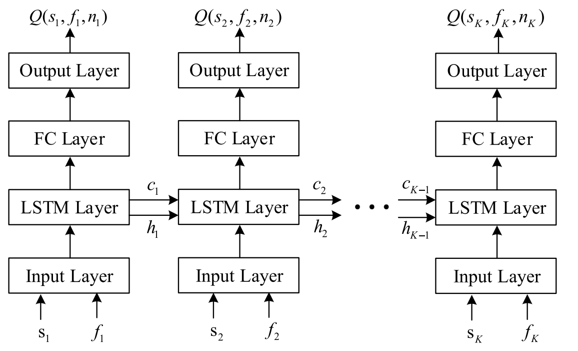


FIGURE 5. The architecture of *critic* network.

Fig.5 illustrates the neural network architecture of the *critic* network, which is similar to the architecture of previous *actor* network. The differences between *critic* network and *actor* network are mainly in input and output aspects. The input of *critic* network includes state and learned computation frequency from *actor* network. The output of *critic* network is the Q values with respect to computation nodes.

We summary the DRL-based task offloading and resource scheduling algorithm (DTORS) in Alg.2. To balance the exploration and exploitation tradeoff, we utilize the ε-greedy policy to select the computation nodes. Ornstein-Uhlenbeck (OU) random noise $\mathcal{O}_k$ is added to actor policy for exploration.

Algorithm 2 DRL-Based Task Offloading and Resource Scheduling Algorithm (DTORS)

```
Input: $G, w_k, d_{j,k}, h^u_{j,k}, h^d_{j,k}, P_l, P_a, W_u, W_d, n_0, R_w, F^n_k, \eta^a, \eta^r, \varphi^e, \varphi^t, \varphi^c, \forall j \in$ pred$(k)$
Output: x, f
    Initialize actor network $\mu(s|\theta^\mu)$ and critic network $Q(s, f, n|\theta^Q)$ with weights $\theta^\mu$ and $\theta^Q$.
    Initialize target actor network $\mu(s|\theta^{\mu'})$ and critic network $Q(s, f, n|\theta^{Q'})$ with weights $\theta^{\mu'} \leftarrow \theta^\mu$ and $\theta^{Q'} \leftarrow \theta^Q$.
    Initialize replay buffer $\Re$ and OU random process $\mathcal{O}$.
1:  for episode = 1 to $\Gamma$ do
2:      Receive initial observation state $s_1$.
3:      for $k = 1$ to $K$ do
4:          Obtain the offloading decision of task $k$: $f_k = \mu(s_k|\theta^\mu) + \mathcal{O}_k$ according to the current actor policy $\mu(s_k|\theta^\mu)$ and exploration noise $\mathcal{O}_k$.
5:          With probability $\varepsilon$ select a random $n_k$, otherwise select $n_k = arg \max_n Q(s, f, n|\theta^Q)$. Calaulate $x^n_k$ by (41).
6:          Execute action $n_k, f_k$, obtain reward $r_k$ and next state $s_{k+1}$.
7:          Store transition $(s_k, f_k, n_k, r_k, s_{k+1})$ in $\Re$.
8:          Sample a random mini-batch of $M$ transitions from $\Re$.
9:          Update actor network and critic network by (44) and (47).
10:         Update the target networks by (48) and (49).
11:     end for
12: end for
13: return x, f
```

VI. SIMULATION RESULTS

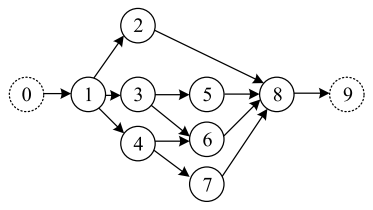
In this section, we first evaluate the convergence performance of proposed STORS algorithm and DTORS algorithm under various system parameters, then four task offloading and resource scheduling schemes are compared with the proposed algorithms in terms of ETRC performance.

A. SIMULATION SETUP

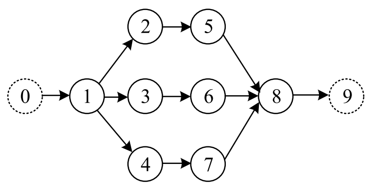
We simulate a hybrid edge-cloud network with 20 SMDs, 4 edge clouds and a remote cloud. The ETRC performance is evaluated by calculating the average ETRC of all SMDs. SMDs are randomly distributed in the coverage of edge clouds of which the coverage radiuses are 100 meters [35]. SMDs walk randomly in these circle regions and the distances between SMDs and edge clouds vary randomly from 10 meters to 100 meters. The initial locations of SMDs are uniformly distributed in these circle coverage regions. The moving direction is uniformly drawn from $[-\pi, \pi]$ at each time slot. The moving velocity of each SMD is updated by the Gaussian Markov mobility model [35]: $v_{t+1} = \rho v_t + (1-\rho)\bar{v} + \delta\sqrt{1-\rho^2}\phi$, where v_t is the moving velocity at time slot t, $\rho \in [0, 1]$ indicates the memory level, $\bar{v} \in \{2, 3, 4, 5, 6\}m/s$ and $\delta = 0.2$ is the mean value and standard variation of velocity, ϕ follows Gaussian distribution $\mathcal{N}(0, 1)$. We generate moving trajectories for SMDs based on the Gaussian Markov mobility model.

The uplink bandwidth W_u and downlink bandwidth W_d are 2 MHz. The maximum transmit powers of SMDs and APs are 100 mW and 1 W [24], respectively. The path-loss model from SMDs to edge clouds is assumed to be $127+30\log_{10} d$, where d is in kilometers. The small-scale channel power gains are modeled as independent and identically distributed Rayleigh fading with unit mean value. The power spectral density $n_0 = -174$ dBm/Hz [36]. Considering the long propagation distances between edge clouds and remote cloud, the average network throughput R_w assigned to each SMD is 5Mbps [7].

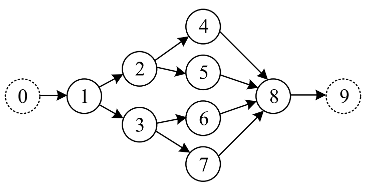
The maximal computation resources of SMDs, edge clouds, and remote cloud are 2 GHz, 20GHz, 100GHz, respectively. The CPU cycles w_k (Mega cycles) and data size $d_{j,k}$ (KB) of tasks follow Gaussian distributions $\mathcal{N}(\mu_c, 100)$ and $\mathcal{N}(\mu_d, 50)$ [3]. The unit price η^r is 0.9×10^{-10} [37]. The computing efficiency parameter $\kappa = 10^{-27}$ [36]. Without losing generality, the ratio of φ^e and φ^t is usually set to 1:1 [3], [37]. The value of φ^c is 1 to 5 times that of φ^e [37]. In the simulations, we first set $\varphi^e = 1, \varphi^t = 1, \varphi^c = 5$ and evaluate the ETRC performance under various network parameters.



(a) The general task call graph.

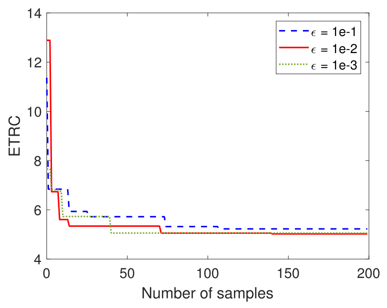


(b) The mesh task call graph.

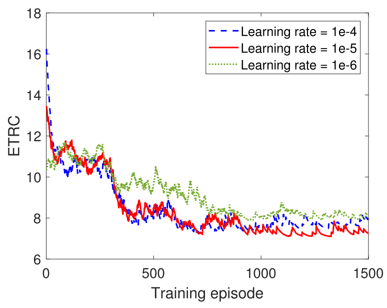


(c) The tree task call graph.

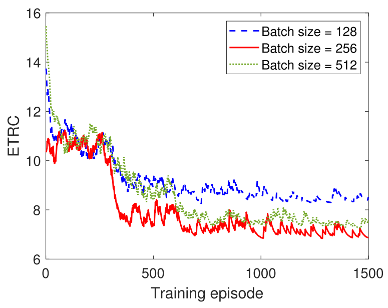
FIGURE 6. Three task call graphs in the simulations.



(a) The ETRC v.s. stopping criterion ϵ.



(b) The ETRC v.s. learning rate ε.



(c) The ETRC v.s. batch size M.

FIGURE 7. The ETRC v.s. stopping criterion ϵ,learning rate ε, and batch size M.

In the proposed DTORS algorithm, the LSTM layer and FC layer have two hidden layers and three hidden layers, respectively. Each hidden layer has 128 neurons. We use Adam optimizer and set the learning rate as 10^{-5}. The training batch size is 256 and the memory size is 10^6. All simulations are implemented in Python with TensorFlow and performed on a computer with an Intel Core i7-9700, 3.0GHz processor and 32GB RAM memory.

B. CONVERGENCE PERFORMANCE

We first evaluate the convergence performance of proposed STORS algorithm and DTORS algorithm. The application consists of 8 tasks with the general task call graph as shown in Fig. 6(a).

In Fig.7(a), we plot the varieties of ETRC with the number of samples V in STORS algorithm under different iteration stopping criterions ϵ of Alg.1. When ϵ is large, we need to sample more random variables from multivariate Gaussian distribution to recover task offloading decisions. If a small ϵ is used, the ETRC can be converged with a fewer number of samples. For instance, the ETRC converges to a stable value after 40 samples when $\epsilon = 10^{-3}$. When $\epsilon = 10^{-1}$, it takes at least 110 samples to reach a stable ETRC. However, too small ϵ increases the iteration time for each sampling. In the following simulations, we set the ϵ as 10^{-2} in proposed STORS algorithm.

Fig.7(b) and Fig.7(c) illustrate the evolution of ETRC under DTORS algorithm with different learning rates ε and batch sizes M. It is observed that the DTORS algorithm can converge to stable values under various learning rates and batch sizes. However, a too large learning rate or a too small learning rate leads to higher fluctuation and converges to a high ETRC. Hence, we set the learning rate as 10^{-5} in the following simulations. Fig.7(c) shows the convergence performance under three batch sizes. When the batch size is 128, the DTORS algorithm converges to a high ETRC. When the batch size is 512, the convergence speed of DTORS algorithm is slower than that with $M = 256$. We set the batch size as 256 in the proposed DTORS algorithm.

C. COMPARISONS WITH OTHER TASK OFFLOADING AND RESOURCE SCHEDULING SCHEMES

In this section, we compare the proposed STORS and DTORS algorithms with four task offloading and resource scheduling schemes: Exhaustive search, Cloud only, Edge only, and Local only. Each SMD has an application with a random task call graph as shown in Fig.6. These three task call graphs are also used in previous work [24]. The ETRC performance is evaluated under various system parameters including the deadline of applications $T_{\max}$, the mean value of data size μ_d, the mean value of workload μ_c, the unit price η^a of edge cloud, the weights ψ_e, ψ_t, ψ_c, and the number of SMDs. The baseline algorithms are given as follows.

1) EXHAUSTIVE SEARCH

We enumerate all feasible task offloading decisions for each application. For each offloading decision, the corresponding computation frequencies are obtained according to

(29), (32), and (37). Then, we select the optimal offloading decision and computation frequency which minimize the ETRC. The Exhaustive search method achieves the optimal ETRC and provides a lower bound for analyzing the performance of other algorithms.

2) CLOUD ONLY

The SMDs' applications are offloaded to remote cloud and the total cost of each SMD is $\varphi^e E_u + \varphi^t T + \varphi^c C$. The computation frequencies of cloud execution are obtained by minimizing $\varphi^e E_u + \varphi^t T + \varphi^c C$.

3) EDGE ONLY

The applications are executed at edge clouds and the total cost of each SMD is $\varphi^e E_u + \varphi^t T + \varphi^c C$. The computation frequencies of edge execution are scheduled with minimal $\varphi^e E_u + \varphi^t T + \varphi^c C$.

4) LOCAL ONLY

The applications are executed at SMDs and the total cost includes the energy cost and the delay cost of local computing, i.e., $\varphi^e E_l + \varphi^t T$. The computation frequencies of local execution are scheduled with minimal $\varphi^e E_l + \varphi^t T$.

First, we evaluate the ETRC performance versus the deadline of applications with $\mu_c = 600$ Mcycles and $\mu_d = 200$ KB in Fig.8. With the increase of deadline, the ETRC becomes lower, this is because the computation frequencies leased by SMDs decrease. It is also observed that the proposed DTORS algorithm achieves lower ETRC in comparison to other baseline algorithms, e.g., 17.8%, 25.3% and 43.1% lower ETRC than the Edge only, Cloud only, Local only schemes when $T_{\max}$ is 3 seconds. Besides, the proposed STORS algorithm achieves near optimal performance with full system information. Hence, the proposed algorithms can be applied to various applications with different deadlines.

Fig.9 shows the varieties of ETRC with the mean value of data size under different algorithms with $\mu_c = 600$ Mcycles, $T_{\max} = 3$. The tasks with larger data size have longer transmission time when offloaded to edge clouds and remote cloud. Due to the limited backhaul between APs and remote cloud, the ETRC obtained by Cloud only scheme increases linearly with the mean value of data size. The uplink transmission rate is much faster than the backhaul rate, thus the ETRC of Edge only scheme has little growth with the increase of data size. The ETRC of proposed STORS algorithm and DTORS algorithm increases slowly and can be applied to different applications with various data sizes. For instance, when the mean value of data size is 200 Kbytes, the proposed STORS and DTORS algorithms decrease ETRC by at least 43.3% and 21.7% compared with Cloud only, Edge only, and Local only schemes.

We also evaluate the ETRC performance versus the mean value of task workload with $\mu_d = 200$ KB, $T_{\max} = 3$ in Fig.10. When task workload is less than 400 Mcycles, Local only scheme has less ETRC than Cloud only and Edge only schemes. When task workload increases with task workload, the ETRC under Local only scheme increases rapidly and becomes larger than that under Cloud only and Edge only

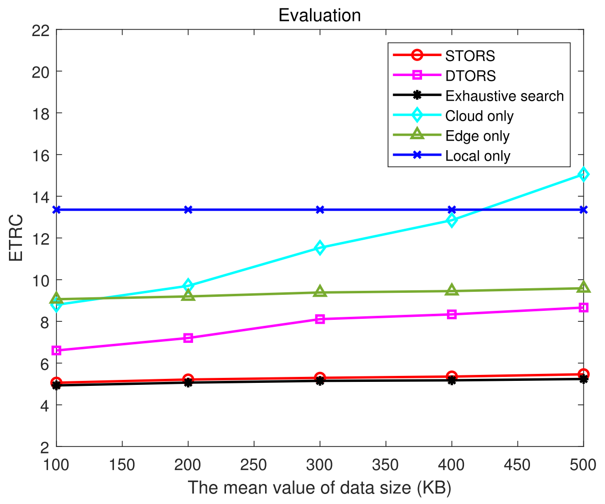


FIGURE 9. The ETRC v.s. the mean value of data size μ_d.

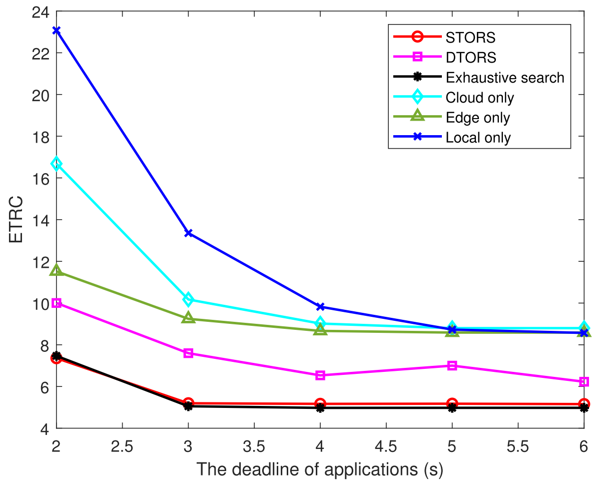


FIGURE 8. The ETRC v.s. deadline $T_{\max}$.

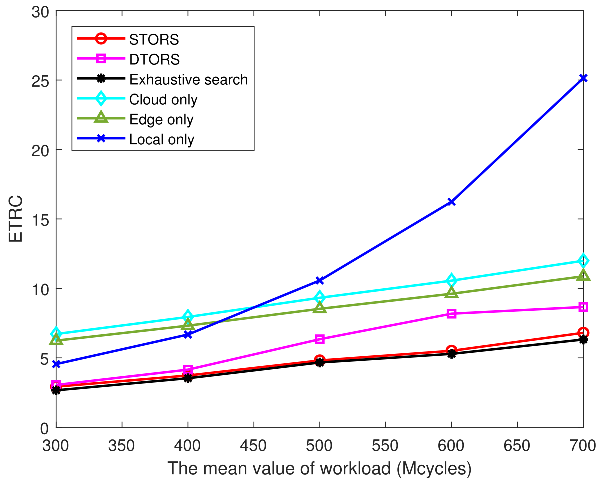


FIGURE 10. The ETRC v.s. the mean value of workload μ_c.

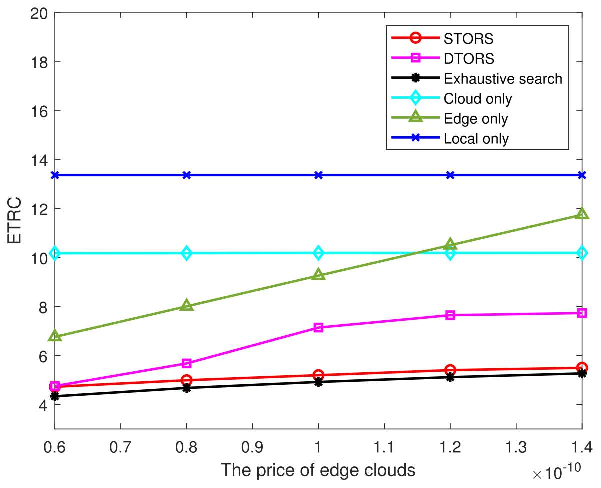


FIGURE 11. The ETRC v.s. price of edge cloud η^a.

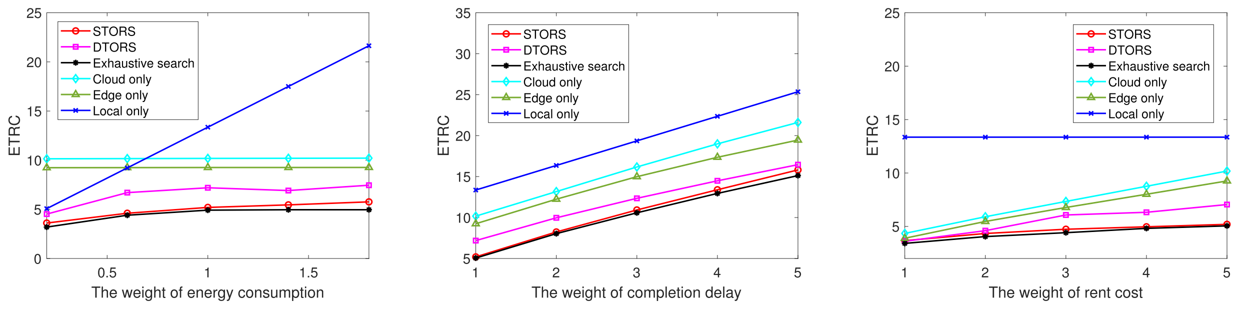
schemes. This is because the energy consumption of SMD grows linearly with task workload and increases at a quadratic rate with computation frequency. The proposed STORS and DTORS algorithms utilize the dependency among tasks and have lower ETRC compared with Cloud only, Edge only, and Local only schemes.

We compare the ETRC performance under various rent prices of edge clouds with $\mu_c = 600$ Mcycles, $\mu_d = 200$ KB, $T_{\max} = 3$ in Fig.11. It is observed that with the increase of rent price η^a, the ETRC with Edge only scheme first increases linearly. As for DTORS algorithm, the ETRC first increases and then remains stable when $\eta^a > 1.2 \times 10^{-10}$. That is because under a large rent price η^a, the computation resources leased from edge clouds reduce and part of tasks executed at edge clouds are offloaded to the remote cloud. Moreover, the proposed STORS and DTORS algorithms can significantly decrease the ETRC compared with baseline schemes. When the price of edge clouds is 1.0×10^{-10}, the STORS algorithm and DTORS algorithm reduce at least 43.9%, and 38.7% lower ETRC than that with Edge only, Cloud only and Local only schemes.

In Fig. 12, we evaluate the ETRC performance versus the weight of energy, time, and rent cost with $T_{\max} = 3$. First, Fig.12(a) illustrates the ETRC performance under different weights of energy consumption φ^e. It can be found that the ETRC under Local only scheme increases rapidly with φ^e, whereas the ETRC under Cloud only scheme or Edge only scheme is almost unchanged. This is because the energy consumption of local computing is much more than that of data transmission. Under efficient task offloading and resource scheduling schemes, the ETRC under STORS algorithm or DTORS algorithm increases slowly and remains stable as the increase of φ^e. The proposed STORS and DTORS algorithms achieve at least 43.9% and 22.3% lower ETRC than Edge only, Cloud only and Local only schemes when $\varphi^e = 1$. In Fig.12(b), we evaluate the performance of ETRC versus the weight of completion delay φ^t. In comparison with the Exhaustive search method, the proposed STORS algorithm achieves near-optimal ETRC. The proposed STORS and DTORS algorithms decrease at least 18.8% and 15.5% ETRC than Cloud only, Edge only, and Local only schemes when $\varphi^t = 5$. Fig.12(c) depicts the comparison results of ETRC versus φ^c. We can observe that the ETRC increases with φ^c under the proposed algorithms, Cloud only scheme, and Edge only scheme. The growth rates of ETRC under Cloud only and Edge only schemes are higher than that under proposed STORS and DTORS algorithms. This is because the proposed algorithms can reduce the number of tasks offloaded to clouds and decrease the computation resources rented from clouds when φ^c is large. Since the computation resources are not rented from clouds when tasks are executed at SMDs, the ETRC keeps unchanged with φ^c under Local only scheme.

The weights φ^e, φ^t and φ^c indicate the preferences of SMDs for energy, delay and price. In Fig. 13, we plot the variations of energy consumption, completion time, and rent cost with weights φ^e, φ^t and φ^c, respectively. We can observe that energy consumption, completion time and rent cost decrease as the increase of φ^e, φ^t and φ^c. If an SMD prefers lower energy consumption, completion time and monetary cost, larger φ^e, φ^t and φ^c should be selected respectively. For instance, if SMDs prefer to complete applications, as shown in Fig.6, by consuming no more than 1 Joule of energy, the weight φ^e should be larger than 1.0 according to the first subfigure of Fig.13.

Furthermore, to verify the superiority of proposed algorithms under various network scales, we compare the



(a) The ETRC v.s. the weight φ^e. (b) The ETRC v.s. the weight φ^t. (c) The ETRC v.s. the weight φ^c.

FIGURE 12. The ETRC v.s. the weight of energy, time and rent cost.

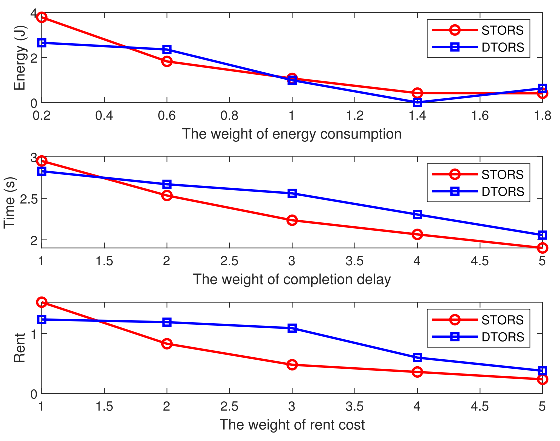


FIGURE 13. **The energy, time, rent v.s. φ^e, φ^t, φ^c.**

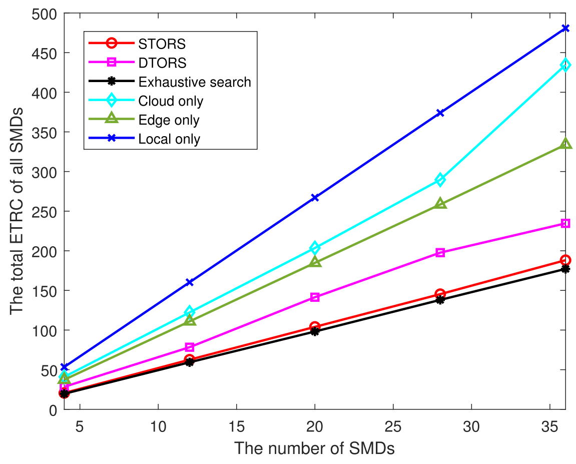


FIGURE 14. **The total ETRC v.s. the number of SMDs.**

proposed algorithms with baseline schemes under different numbers of SMDs in Fig.14. We can observe that the total ETRC of all SMDs increases with the number of SMDs. The total ETRC of proposed STORS algorithm and DTORS algorithm increases more slowly than that of Cloud only, Edge only and Local only schemes. This is reasonable since the proposed algorithms can schedule tasks in parallel according to the interdependency of tasks. The growth rate of ETRC under the Cloud only scheme becomes faster when the number of SMDs is larger than 28. This is due to the long transmission delay between SMDs and remote cloud, SMDs need to rent more computation resources from remote cloud to satisfy deadline constraints. The simultaneous resource requests make remote cloud unable to schedule enough computation resources for each SMD and some SMDs acquire fewer computation resources, which makes the total ETRC grow faster.

VII. CONCLUSION

This paper investigated the task offloading and resource scheduling problem for inter-dependent tasks in hybrid edge-cloud networks. Under the conditions of full system information and partial system information, we proposed two efficient task offloading and resource scheduling algorithms to minimize the weighted sum of energy, time, and rent cost for each SMD. With full system information, the task offloading decisions were obtained by semidefinite relaxation and Gaussian randomization sampling. The corresponding computation frequencies were derived based on the dual decomposition method. With partial system information, a DRL framework was designed to learn the discrete task offloading decisions and continuous computation frequencies. We have improved the conventional DDPG algorithm by redesigning the *actor* network and *critic* network so as to learn the discrete action and continuous action simultaneously. Furthermore, we introduced LSTM layers into *actor* network and *critic* network to better predict the time series system states. Simulation results have demonstrated the convergence of proposed algorithms on different system parameters. Extensive simulations also verified the proposed algorithms have achieved near-optimal performance and significantly decreased the ETRC compared with baselines.

For the future work, we will consider the competition among multiple SMDs about bandwidth and computing resources and improve the ETRC performance by multi-agent reinforcement learning techniques. In addition, most existing studies assume that edge clouds have deployed all applications requested by SMDs. However, considering the limited storage capacities of edge clouds, it is interesting to investigate the service caching and task offloading scheme for mobile edge computing. Another direction is to consider the wireless interference among SMDs and investigate the joint bandwidth, transmit power and computation resource scheduling for hybrid edge-cloud networks.

REFERENCES

[1] Y. Liu, M. Peng, G. Shou, Y. Chen, and S. Chen, "Toward edge intelligence: Multiaccess edge computing for 5G and Internet of Things," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 6722–6747, Aug. 2020.

[2] Y. Zhang, X. Lan, J. Ren, and L. Cai, "Efficient computing resource sharing for mobile edge-cloud computing networks," *IEEE/ACM Trans. Netw.*, vol. 28, no. 3, pp. 1227–1240, Jun. 2020.

[3] S. Guo, J. Liu, Y. Yang, B. Xiao, and Z. Li, "Energy-efficient dynamic computation offloading and cooperative task scheduling in mobile cloud computing," *IEEE Trans. Mobile Comput.*, vol. 18, no. 2, pp. 319–333, Feb. 2019.

[4] 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, 4th Quart., 2017.

[5] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.

[6] J. Ren, G. Yu, Y. He, and G. Y. Li, "Collaborative cloud and edge computing for latency minimization," *IEEE Trans. Veh. Technol.*, vol. 68, no. 5, pp. 5031–5044, May 2019.

[7] J. Liu, K. Xiong, D. W. K. Ng, P. Fan, Z. Zhong, and K. B. Letaief, "Max-min energy balance in wireless-powered hierarchical fog-cloud computing networks," *IEEE Trans. Wireless Commun.*, vol. 19, no. 11, pp. 7064–7080, Nov. 2020.

[8] T. Q. Dinh, B. Liang, T. Q. S. Quek, and H. Shin, "Online resource procurement and allocation in a hybrid edge-cloud computing system," *IEEE Trans. Wireless Commun.*, vol. 19, no. 3, pp. 2137–2149, Mar. 2020.

[9] T. Cao, C. Xu, J. Du, Y. Li, H. Xiao, C. Gong, L. Zhong, and D. Niyato, "Reliable and efficient multimedia service optimization for edge computing-based 5G networks: Game theoretic approaches," *IEEE Trans. Netw. Service Manage.*, vol. 17, no. 3, pp. 1610–1625, Sep. 2020.

[10] Y. Dai, K. Zhang, S. Maharjan, and Y. Zhang, "Edge intelligence for energy-efficient computation offloading and resource allocation in 5G beyond," *IEEE Trans. Veh. Technol.*, vol. 69, no. 10, pp. 12175–12186, Oct. 2020.
[11] F. Fang, Y. Xu, Z. Ding, C. Shen, M. Peng, and G. K. Karagiannidis, "Optimal resource allocation for delay minimization in NOMA-MEC networks," *IEEE Trans. Commun.*, vol. 68, no. 12, pp. 7867–7881, Dec. 2020.
[12] J. Zhang, J. Du, Y. Shen, and J. Wang, "Dynamic computation offloading with energy harvesting devices: A hybrid-decision-based deep reinforcement learning approach," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9303–9317, Oct. 2020.
[13] Q. Huang, P. Ang, P. Knowles, T. Nykiel, I. Tverdokhlib, A. Yajurvedi, P. Dapolito, X. Yan, M. Bykov, C. Liang, M. Talwar, A. Mathur, S. Kulkarni, M. Burke, and W. Lloyd, "SVE: Distributed video processing at facebook scale," in *Proc. 26th Symp. Operating Syst. Princ.*, New York, NY, USA, Oct. 2017, pp. 87–103.
[14] C. Shu, Z. Zhao, Y. Han, G. Min, and H. Duan, "Multi-user offloading for edge computing networks: A dependency-aware and latency-optimal approach," *IEEE Internet Things J.*, vol. 7, no. 3, pp. 1678–1689, Mar. 2020.
[15] L. Liu, H. Tan, S. H.-C. Jiang, Z. Han, X.-Y. Li, and H. Huang, "Dependent task placement and scheduling with function configuration in edge computing," in *Proc. Int. Symp. Qual. Service*, Jun. 2019, pp. 1–10.
[16] Q. Wang, L. T. Tan, R. Q. Hu, and Y. Qian, "Hierarchical energy-efficient mobile-edge computing in IoT networks," *IEEE Internet Things J.*, vol. 7, no. 12, pp. 11626–11639, Dec. 2020.
[17] S. Ahn, J. Lee, T. Y. Kim, and J. K. Choi, "A novel edge-cloud interworking framework in the video analytics of the Internet of Things," *IEEE Commun. Lett.*, vol. 24, no. 1, pp. 178–182, Jan. 2020.
[18] G. Peng, H. Wu, H. Wu, and K. Wolter, "Constrained multi-objective optimization for IoT-enabled computation offloading in collaborative edge and cloud computing," *IEEE Internet Things J.*, early access, Mar. 22, 2021, doi: 10.1109/JIOT.2021.3067732.
[19] I. Kovacevic, E. Harjula, S. Glisic, B. Lorenzo, and M. Ylianttila, "Cloud and edge computation offloading for latency limited services," *IEEE Access*, vol. 9, pp. 55764–55776, 2021.
[20] 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.
[21] Y. Geng, Y. Yang, and G. Cao, "Energy-efficient computation offloading for multicore-based mobile devices," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, Apr. 2018, pp. 46–54.
[22] S. Zhu, L. Gui, D. Zhao, N. Cheng, Q. Zhang, and X. Lang, "Learning-based computation offloading approaches in UAVs-assisted edge computing," *IEEE Trans. Veh. Technol.*, vol. 70, no. 1, pp. 928–944, Jan. 2021.
[23] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading tasks with dependency and service caching in mobile edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 11, pp. 2777–2792, Nov. 2021.
[24] 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.
[25] K. Zhang, S. Leng, Y. He, S. Maharjan, and Y. Zhang, "Mobile edge computing and networking for green and low-latency Internet of Things," *IEEE Commun. Mag.*, vol. 56, no. 5, pp. 39–45, May 2018.
[26] A. Agrawal, R. Verschueren, S. Diamond, and S. Boyd, "A rewriting system for convex optimization problems," *J. Control Decis.*, vol. 5, no. 1, pp. 42–60, Jan. 2018.
[27] Z.-Q. Luo, W.-K. Ma, A. So, Y. Ye, and S. Zhang, "Semidefinite relaxation of quadratic optimization problems," *IEEE Signal Process. Mag.*, vol. 27, no. 3, pp. 20–34, May 2010.
[28] S. Boyd, L. Xiao, and A. Mutapcic. (2003). Subgradient methods. Stanford Univ., Stanford, CA, USA, Autumn Quarter, Lect. Notes EE392o, pp. 2004–2005.
[29] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.
[30] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *Proc. Int. Conf. Learn. Represent. (ICRL)*, May 2016, pp. 1–14.
[31] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2014, pp. 387–395.
[32] M. Tang and V. W. S. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Trans. Mobile Comput.*, early access, Nov. 10, 2020, doi: 10.1109/TMC.2020.3036871.
[33] Z. Yang, Y. Liu, Y. Chen, and N. Al-Dhahir, "Cache-aided NOMA mobile edge computing: A reinforcement learning approach," *IEEE Trans. Wireless Commun.*, vol. 19, no. 10, pp. 6899–6915, Oct. 2020.
[34] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[35] J. Li, X. Zhang, J. Zhang, J. Wu, Q. Sun, and Y. Xie, "Deep reinforcement learning-based mobility-aware robust proactive resource allocation in heterogeneous networks," *IEEE Trans. Cognit. Commun. Netw.*, vol. 6, no. 1, pp. 408–421, Mar. 2020.
[36] Q. Zhang, L. Gui, F. Hou, J. Chen, S. Zhu, and F. Tian, "Dynamic task offloading and resource allocation for mobile-edge computing in dense cloud RAN," *IEEE Internet Things J.*, vol. 7, no. 4, pp. 3282–3299, Apr. 2020.
[37] S.-H. Kim, S. Park, M. Chen, and C.-H. Youn, "An optimal pricing scheme for the energy-efficient mobile edge computation offloading with OFDMA," *IEEE Commun. Lett.*, vol. 22, no. 9, pp. 1922–1925, Sep. 2018.

QI ZHANG received the B.Eng. degree from the School of Electronics and Information, Northwestern Polytechnical University (NPU), Xi'an, China, in 2015. He is currently pursuing the Ph.D. degree with the Department of Electronic Engineering, Shanghai Jiao Tong University (SJTU), Shanghai, China. His research interests include mobile edge computing, resource management, and optimization.

LIN GUI (Member, IEEE) received the Ph.D. degree from Zhejiang University, Hangzhou, China, in 2002. Since 2002, she has been with the Institute of Wireless Communication Technology, Shanghai Jiao Tong University, Shanghai, China, where she is currently a Professor. Her current research interests include HDTV and wireless communications.

SHICHAO ZHU received the B.E. degree from the School of Aeronautics, Northwestern Polytechnical University, Xi'an, China, in 2016. He is currently pursuing the Ph.D. degree with the Department of Electronic Engineering, Shanghai Jiao Tong University, Shanghai, China. His current research interests include data and computation offloading in UAV networks and space-air integrated networks, and the application of AI in wireless networks.

XIUPU LANG received the B.E. degree from the School of Electronics and Information Engineering, Harbin Institute of Technology, Harbin, China, in 2018. He is currently pursuing the Ph.D. degree with the Department of Electronic Engineering, Shanghai Jiao Tong University, Shanghai, China. His current research interests include software defined network design, network virtualization, and resources management.