Offloading and Resource Allocation With General Task Graph in Mobile Edge Computing: A Deep Reinforcement Learning Approach

Jia Yan, *Student Member, IEEE*, Suzhi Bi, *Senior Member, IEEE*, and Ying-Jun Angela Zhang, *Fellow, IEEE*

***Abstract*— In this paper, we consider a mobile-edge computing (MEC) system, where an access point (AP) assists a mobile device (MD) to execute an application consisting of multiple tasks following a general task call graph. The objective is to jointly determine the offloading decision of each task and the resource allocation (e.g., CPU computing power) under time-varying wireless fading channels and stochastic edge computing capability, so that the energy-time cost (ETC) of the MD is minimized. Solving the problem is particularly hard due to the combinatorial offloading decisions and the strong coupling among task executions under the general dependency model. Conventional numerical optimization methods are inefficient to solve such a problem, especially when the problem size is large. To address the issue, we propose a deep reinforcement learning (DRL) framework based on the actor-critic learning structure. In particular, the actor network utilizes a DNN to learn the optimal mapping from the input states (i.e., wireless channel gains and edge CPU frequency) to the binary offloading decision of each task. Meanwhile, by analyzing the structure of the optimal solution, we derive a low-complexity algorithm for the critic network to quickly evaluate the ETC performance of the offloading decisions output by the actor network. With the low-complexity critic network, we can quickly select the best offloading action and subsequently store the state-action pair in an experience replay memory as the training dataset to continuously improve the action generation DNN. To further reduce the complexity, we show that the optimal offloading decision exhibits an one-climb structure, which can be utilized to significantly reduce the search space of action generation. Numerical results show that for various types of task graphs, the proposed algorithm achieves up to 99.1% of the optimal performance while significantly reducing the computational complexity compared to the existing optimization methods.**

***Index Terms*— Mobile edge computing, optimization algorithm, deep reinforcement learning, resource allocation.**

Manuscript received October 15, 2019; revised January 18, 2020 and March 10, 2020; accepted April 24, 2020. Date of publication May 14, 2020; date of current version August 12, 2020. This work was supported in part by the National Natural Science Foundation of China under Project 61871271, in part by the General Research Fund established by the Research Grants Council of Hong Kong under Project 14208017, in part by the Guangdong Province Pearl River Scholar Funding Scheme 2018 under Project 308/00003704, in part by the Foundation of Shenzhen City under Project JCYJ20170818101824392 and Project JCYJ20190808120415286, and in part by the Science and Technology Innovation Commission of Shenzhen under Project 827/000212. This article will be presented in part at the IEEE International Conference on Communications, Dublin, Ireland, June 7–11, 2020 [1]. The associate editor coordinating the review of this article and approving it for publication was C. Huang. *(Corresponding author: Suzhi Bi.)*

Jia Yan and Ying-Jun Angela Zhang are with the Department of Information Engineering, The Chinese University of Hong Kong, Hong Kong (e-mail: yj117@ie.cuhk.edu.hk; yjzhang77@ieee.org).

Suzhi Bi is with the College of Electronic and Information Engineering, Shenzhen University, Shenzhen 518060, China (e-mail: bsz@szu.edu.cn).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TWC.2020.2993071

I. Introduction

RECENT years have witnessed explosive growth of Internet of Things (IoT) as a way to connect tens of billions of resource-limited wireless devices, such as sensors, mobile devices (MDs) and wearable devices, to Internet through the cellular networks. Due to small physical sizes and stringent production costs constraints, IoT devices often suffer from limited computation capabilities and finite battery lives. Perceived as a promising solution, mobile edge computing (MEC) [2], [3] has attracted significant attention. With MEC, computationally intensive tasks can be offloaded to nearby servers located at the edges of wireless networks. This efficiently overcomes the drawbacks of long backhaul latency and high overhead compared to traditional mobile cloud computing.

Typically, there are two computation task offloading models for MEC [2]: one is referred to as binary offloading, and the other is partial offloading. For the binary offloading model, each task is either executed locally or offloaded to the MEC server as a whole [4]–[9]. As for partial offloading, tasks can be arbitrarily divided into two parts that are executed by the device and the edge server, respectively [10], [11]. Nevertheless, in practice, a mobile application usually has multiple components and the dependency among them cannot be ignored since the outputs of some components are the inputs of others. In this regard, task call graph [12] is proposed to model the sophisticated inter-dependency among different components in a mobile application. In this paper, we consider computation offloading with a general task call graph.

Due to the random variation of wireless channels, it is not always advantageous to offload all the tasks for edge execution. Instead, offloading computation tasks in an opportunistic manner considering the time-varying channel condition has shown significant performance advantage [4]–[11]. Due to the mutual coupling constraints in a task call graph, offloading policy design becomes much challenging [13]–[18]. Specifically, [13] considered a sequential task graph and derived an optimal one-climb policy, where the execution migrates

only at most once between the MD and the cloud server. This work was extended to a general task graph case in [14], where authors applied the partial critical path analysis for the general task graph scheduling. In [15], the offloading problem in a general task graph was formulated as a linear programming problem through convex relaxation. Reference [16] modeled the task scheduling problem in a general task graph as an energy consumption minimization problem that is solved by a genetic algorithm. Note that general task graphs are considered much harder to deal with compared to other task graphs with special structures (i.e., sequential task graph), since it is hard to explore and derive the offloading properties (i.e., one-climb policy in the sequential task graph) with the general and complicated coupling among tasks.

On the other hand, recent work has considered joint optimization of radio/computing resource allocation and computation offloading. In particular, [17] studied an energy-efficiency cost minimization problem by incorporating CPU frequency control and transmit power allocation in the MEC offloading decision. Reference [18] considered inter-user task dependency and proposed a reduced-complexity Gibbs sampling algorithm to obtain the optimal offloading decisions.

The existing work on task offloading with general task graph adopts either convex relaxation methods (e.g., in [15], [17]) or heuristic local search methods (e.g., in [13], [14], [16], [18]). However, both methods are likely to get stuck in a local optimal solution that does not guarantee good performance. Moreover, the optimization problems need to be re-solved once the wireless channel conditions change or the available computing power of the edge server changes due to the variation of demands by background applications. The frequent re-calculation of offloading decisions renders the existing methods impractical.

In this paper, we endeavor to design an efficient optimal computation offloading algorithm in an MEC system with a general task graph, so that the optimal decision swiftly adapts to the time-varying wireless channels and available edge computing power with very low computational complexity. In particular, we propose a deep reinforcement learning (DRL) framework. The key idea of DRL is to utilize the deep neural networks (DNNs) to learn the optimal mapping between the state space and the action space. There exists several work on DRL-based offloading methods for MEC systems [19]–[21]. In [19], a deep Q-network (DQN) based offloading policy was proposed to optimize the computational performance in the MEC system with energy harvesting. When tasks arrive randomly, [20] proposed DQN to learn the optimal offloading decisions without a priori knowledge of network dynamics. To tackle the curse of dimensionality problem in DQN-based methods, [21] proposed a novel DRL framework to achieve near-optimal offloading actions by considering only a small subset of candidate offloading actions in each iteration. Notice that [19]–[21] all assume independent tasks among multiple users. Very recently, considering a general task dependency, [22] proposed a recurrent neural network (RNN) based reinforcement learning method for the computation offloading problem. However, it neglected the system dynamics, such as wireless fading channels and time-varying edge server CPU frequency.

We consider an MEC system with a single access point (AP) and a MD as shown in Fig. 1. The MD has an application with a general task topology to execute under time-varying wireless fading channels and edge server CPU frequency. In particular, we propose a DRL framework to minimize the weighted sum of task execution time and energy consumption of the MD. The main contributions are concluded as follows:

- We formulate a mixed integer optimization problem to jointly optimize the offloading decisions and local CPU frequencies of the MD to minimize the computation delay and energy consumption. The problem is challenging because of the combinatorial nature of the offloading decisions and the strong coupling among task executions under general dependency model.
- In order to solve the combinatorial optimization problem efficiently, we propose a DRL framework based on the actor-critic learning structure, where we train a DNN in the actor network periodically from the past experiences to learn the optimal mapping between the states (i.e., wireless channels and edge CPU frequency) and actions (i.e., offloading decisions). Within the actor network, we devise a novel Gaussian noise-added order-preserving action generation method to balance the diversity and complexity in generating candidate binary offloading actions under a high-dimensional action space.
- For the critic network, we simplify the problem according to the total loop-free paths in the general task graph and derive closed-form solution for the optimal local CPU frequencies. Based on this, we propose an efficient algorithm. As such, unlike traditional actor-critic networks that utilize a DNN to predict the values of the actions in the critic network, our analysis allows fast and accurate calculation of the performance of each action generated by the actor network. In this way, the complexity and convergence of the actor-critic based DRL are greatly improved.
- To further speed up the computation of the proposed DRL framework, we propose a heuristics where the offloading decisions are limited to the ones that follow the one-climb offloading policy. The heuristics greatly reduces the number of performance evaluations for the actions in the critic network. The optimality of the one-climb policy is analyzed and its advantageous performance over conventional action generation method is verified through simulations.

Numerical results show that for various types of general task graphs, the proposed DRL-based algorithm achieves up to 99.1% of the optimal energy and time cost. Meanwhile, our proposed method only takes around 1 second to generate an offloading action, which is more than one order of magnitude faster than the other representative benchmark methods. In this paper, we formulate the joint optimization of offloading and resource allocation with general task graph in the MEC as a mixed integer non-linear programming (MINLP) problem, which is hard to solve with conventional optimization

algorithms under time-varying wireless channels and stochastic edge computing capability. By exploring the special structure of the considered MINLP problem, we observe that for any given integer variables (offloading decisions), the remaining problem is convex. Therefore, the main difficulty lies in finding the optimal integer offloading decisions. With such property, we propose the actor-critic learning structure based DRL algorithm, where the actor network generates a set of integer offloading actions according to the time-varying parameters and the critic network scores each action output from the actor network by convex optimization. Then, we utilize the generated action-score pairs to make current offloading decision and improve the performance of the actor network. It is worth mentioning that the key target of the critic is for evaluating the action quality, regardless of using a general neural network or a specialized algorithm [23]. In this paper, as one of the major contributions, we propose an efficient low-complexity algorithm in the critic network to evaluate the actions generated from the actor network, which greatly reduces the training cost of the critic DNN and increases the accuracy of action evaluation.

The rest of the paper is organized as follows. In Section II, we present the system model and problem formulation. The optimal local CPU frequencies under fixed offloading decisions are studied in Section III. We introduce the detailed design for the DRL framework in Section IV. In Section V, simulation results are described. Finally, we conclude the paper in Section VI.

II. System Model and Problem Formulation

As shown in Fig. 1, we consider an MEC system with one AP and one MD. The AP is the gateway of the edge cloud and has stable power supply. The MD has a computationally intensive mobile application consisting of M dependent tasks. The input-output dependency of the tasks is represented by a directed acyclic task graph $G = (\mathcal{M}, \mathcal{E})$. As shown in Fig. 2, each vertex in G represents a task i and the associated parameter L_i indicates the computing workload in terms of the total number of CPU cycles required for accomplishing the task. Besides, each edge $(k, i) \in \mathcal{E}$ in G represents that a precedent task k must be completed before starting to execute task i. Additionally, we denote the size of data in bits transferred from task k to i by $O_{k,i}$. For simplicity of exposition, we introduce two virtual tasks 0 and $M + 1$ as the entry and exit tasks, respectively. Specifically, we have $L_0 = L_{M+1} = 0$. By forcing the two virtual tasks to be executed locally, we ensure that the application is initiated and terminated at the MD side. We denote the set of tasks in the task graph G as $\mathcal{M} = \{0, 1, \ldots, M + 1\}$.

Define an indicator variable $a_i \in \{0, 1\}$ such that $a_i = 0$ means that task i is executed locally and $a_i = 1$ means that the MD offloads the computation of task i to the edge side. Recall that the two virtual tasks 0 and $M+1$ must be executed locally. That is, $a_0 = a_{M+1} = 0$.

In addition, we assume that the MD is allocated a dedicated spectral resource block throughout its transmission, which can support concurrent transmissions for task offloading and downloading. We denote by $h^u_{k,i}$ and $h^d_{k,i}$ the channel gains when

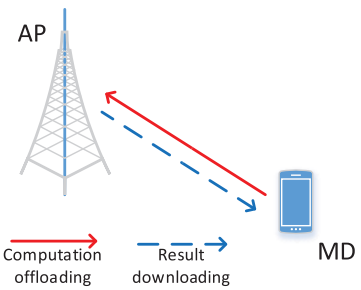


Fig. 1. System model.

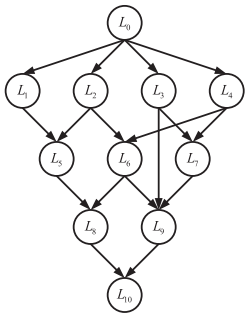


Fig. 2. The considered task graph.

offloading and downloading the task data $O_{k,i}$, respectively. Besides, we assume additive white Gaussian noise (AWGN) with zero mean and equal variance σ^2 at the receiver for all the tasks.

To characterize the task execution time and energy consumption for local and edge computing, respectively, we first define the *finish time* and *ready time* of each task.

Definition 1 (Finish Time): The finish time of task i is the moment when all the workload L_i has been executed. We denote FT^l_i and FT^c_i as the finish time of task i when it is executed locally and at the edge server, respectively.

Definition 2 (Ready Time): The ready time of a task is the earliest time when the task has received all the necessary input data to commence the task computation. For instance, in Fig. 2, the ready time of the fifth task is the time when both the input data streams from the first and second tasks have arrived. We denote the ready time of task i when computing locally and at the edge server as RT^l_i and RT^c_i, respectively.

A. Local Computing

We assume that the MD is equipped with a ρ^l-core CPU, where each CPU core can execute only one task at a time. That is, the MD can execute in total ρ^l tasks simultaneously. Suppose that task i is computed locally. We denote the local CPU frequency for computing the task as f^l_i, which is upper bounded by $f^l_i \leq f_{peak}$. Thus, the local execution time of task i is given by

$$\tau^l_i = \frac{L_i}{f^l_i}, \tag{1}$$

and the corresponding energy consumption is [2]

$$e^l_i = \kappa L_i (f^l_i)^2 = \kappa \frac{L^3_i}{(\tau^l_i)^2}, \tag{2}$$

where κ is the effective switched capacitance depending on the chip architecture. According to the circuit theory [24],

the power consumption of the CPU is approximately proportional to the product of $V_{cir}^2 f_i^l$, where V_{cir} is the circuit supplied voltage. Besides, V_{cir} is approximately linear proportional to the CPU frequency f_i^l when the CPU works at the low voltage limits [25]. Therefore, the energy consumption per CPU cycle is given by $\kappa(f_i^l)^2$. It is worth mentioning that for the two virtual tasks 0 and $M+1$, we have $\tau_0^l = \tau_{M+1}^l = 0$ and $e_0^l = e_{M+1}^l = 0$.

If a task k preceding task i is executed at the edge server, then the output data $O_{k,i}$ must be downloaded to the MD before task i can be executed locally. Denote the fixed downlink transmit power of the AP by P_{AP}. Then, according to the Shannon-Hartley theorem, the downlink data rate from the AP to the MD is

$$R_{k,i}^d = W \log_2 \left(1 + \frac{P_{AP} h_{k,i}^d}{\sigma^2}\right), \tag{3}$$

where W denotes the fixed bandwidth of the orthogonal channels allocated to the MD. The corresponding downlink transmission time for sending the data $O_{k,i}, (k,i) \in \mathcal{E}$, is

$$\tau_{k,i}^d = \frac{O_{k,i}}{R_{k,i}^d}. \tag{4}$$

As such, the ready time RT_i^l of task i is given by

$$RT_i^l = \max_{k \in \textbf{pred(i)}} \left\{(1-a_k) FT_k^l + a_k \left(FT_k^c + \tau_{k,i}^d\right)\right\}, \tag{5}$$

where **pred(i)** denotes the set of immediate predecessors of task i. Specifically, if $a_k = 1$ for a task $k \in$ **pred(i)**, the time until its output data is available at the MD for the execution of task i is equal to its finish time FT_k^c at the edge side plus the downlink transmission time $\tau_{k,i}^d$. Otherwise, if $a_k = 0$, the time until its output data is available at the MD is equal to its local finish time FT_k^l. When all needed data is available at the ready time RT_i^l, the MD locally computes task i with the local execution time τ_i^l in (1), so that the finish time of task i becomes

$$FT_i^l = RT_i^l + \tau_i^l. \tag{6}$$

B. Edge Computing

We denote the fixed transmit power of the MD by P_{MD}. Then, the uplink data rate for offloading the data $O_{k,i}, (k,i) \in \mathcal{E}$, to the AP is

$$R_{k,i}^u = W \log_2 \left(1 + \frac{P_{MD} h_{k,i}^u}{\sigma^2}\right), \tag{7}$$

and the corresponding uplink transmission time is

$$\tau_{k,i}^u = \frac{O_{k,i}}{R_{k,i}^u}. \tag{8}$$

The transmission energy consumption is

$$e_{k,i}^u = \tau_{k,i}^u P_{MD}. \tag{9}$$

We assume that the edge server has ρ^c cores and can compute ρ^c tasks in parallel. The execution time of task i on the AP is given by

$$\tau_i^c = \frac{L_i}{f^c}, \tag{10}$$

where f^c is the fixed service rate of each CPU core. Similarly, we can calculate the ready time of task i executed at the edge server as

$$RT_i^c = \max_{k \in \textbf{pred(i)}} \left\{(1-a_k)\left(FT_k^l + \tau_{k,i}^u\right) + a_k FT_k^c\right\}, \tag{11}$$

and its finish time is

$$FT_i^c = RT_i^c + \tau_i^c. \tag{12}$$

C. Problem Formulation

We assume that both the MD and MEC server have a lot more CPU cores than needed to execute the possibly concurrent tasks in the considered mobile application. As such, we can safely set $\rho^l = \rho^c = \infty$. Besides, it is assumed that the number of available channels is sufficiently large to execute the possibly concurrent data transmissions in the task graph.

From the above discussion, the total time to complete the all tasks is equal to the local finish time of the auxiliary exit task $M+1$, i.e., FT_{M+1}^l. Besides, we can calculate the total energy consumption of the MD by

$$E = \sum_{i=1}^{M} (1-a_i) e_i^l + \sum_{i=1}^{M} \sum_{k \in \textbf{pred(i)}} (1-a_k) a_i e_{k,i}^u, \tag{13}$$

which consists of energy consumed on local computation and task offloading.

In this paper, we consider the energy-time cost (ETC) as the performance metric, which is defined as the weighted sum of the total energy consumption and execution time, i.e.,

$$\eta = \beta_e E + \beta_t FT_{M+1}^l, \tag{14}$$

where $0 < \beta_e < 1$ and $0 < \beta_t < 1$ denote the weights of energy consumption and computation completion time of the MD, respectively. It is assumed that the weights are related by $\beta_t = 1 - \beta_e$.

Evidently, a higher CPU frequency leads to shorter task execution time. Meanwhile, according to (2), the energy consumption per CPU cycle is a quadratic function of the CPU frequency, thus the energy consumption increases with the CPU frequency for executing a task. Because the AP has stable power supply, it can operate with a fixed maximum frequency f^c to minimize the execution delay. However, since the MD is often energy-constrained, we can apply dynamic voltage and frequency scaling (DVFS) technique to tune the local CPU frequency for balancing the performance between energy consumption and execution time. Denoting $\mathbf{a} \triangleq \{a_i\}$ and $\mathbf{f} \triangleq \{f_i^l\}$, $i \in \mathcal{M}$, we aim to minimize the ETC of the MD subject to the peak CPU frequency constraint of the MD, i.e.,

$$\begin{aligned} \text{(P1)} \quad & \min_{(\mathbf{a},\mathbf{f})} \ \eta, \\ & \text{s.t.} \ \ 0 \le f_i^l \le f_{peak}, \\ & \qquad a_i \in \{0,1\}, \forall i \in \mathcal{M}, \end{aligned} \tag{15}$$

where we assume $f^c > f_{peak}$ in this paper. We consider the weighted-sum approach [9,17,18] for a general multi-objective optimization problem. According to the Proposition 3.9 of

[26], for any given positive weights, we can reach an efficient solution of the multi-objective optimization problem by solving Problem (P1). A weakly efficient solution will be obtained if any of the weights is zero. Besides, in order to meet user-specific demands, we allow the MD to choose different weights. For instance, the MD with low battery energy prefers a larger β_e for energy saving, while for the delay-sensitive MD, a larger β_t will be chosen to reduce the execution time.

In general, (P1) is non-convex due to the binary variables $\mathbf{a}$ and the recursive structure of FT^l_{M+1}. In the following section, we first simplify (P1) by exploiting the property of the total task completion time FT^l_{M+1}. Then, we propose an efficient method to obtain the optimal CPU frequencies with a given $\mathbf{a}$.

III. Optimal Resource Allocation Under Fixed Offloading Decisions

A. Problem (P1) Simplification

We denote a path o as an ordered sequence of task indices $\Psi(o) = \{k^o_0, k^o_1, \ldots, k^o_m, \ldots, k^o_{m_o}, k^o_{m_o+1}\}$, $k^o_0 = 0, k^o_{m_o+1} = M+1$, that pass through the general task graph G from the entry task 0 to the exit task $M+1$. Here, m_o is the total number of real tasks in path o. For instance, $\{0, 1, 5, 8, 10\}$ is a path in Fig. 2. There are three real tasks $\{1, 5, 8\}$ in the path. Besides, we denote the set of all loop-free paths as $\mathcal{O}$, which can be obtained by running the K-shortest path routing algorithm on G. Likewise, we denote by $O = |\mathcal{O}|$ the total number of paths. Let T_o denote the total execution time in the o-th path excluding the waiting time for the data inputs from the other paths. Then, we have

$$\begin{aligned} T_o = & \sum_{k^o_m \in \Psi(o)} [(1 - a_{k^o_m})\tau^l_{k^o_m} + a_{k^o_m}\tau^c_{k^o_m}] \\ & + \sum_{k^o_m = k^o_1}^{k^o_{m_o+1}} \Big[a_{k^o_m}(1 - a_{k^o_{m-1}})\tau^u_{k^o_{m-1}, k^o_m} \\ & + (1 - a_{k^o_m}) a_{k^o_{m-1}} \tau^d_{k^o_{m-1}, k^o_m} \Big], \end{aligned} \tag{16}$$

which consists of the total computation and communication delay in path o.

To simplify Problem (P1), we first have the following lemma on FT^l_{M+1}.

Lemma 3.1: $FT^l_{M+1} = \max\{T_1, T_2, \ldots, T_o, \ldots, T_O\}$ holds given any $(\mathbf{a}, \mathbf{f})$.

Proof: Please refer to Appendix A. ∎

Lemma 3.1 indicates that the final completion time is equal to the largest total execution time of all the paths in G. Note that although T_o does not include the time spent on waiting for the task input data from other paths, the largest T_o among all paths is the final completion time.

Due to the one-to-one mapping between f^l_i and τ^l_i in (1), it is equivalent to optimize (P1) over the time allocation τ^l_i. By introducing an auxiliary variable $T_{max} = \max\{T_1, T_2, \ldots, T_o, \ldots, T_O\}$, (P1) can be equivalently expressed as

$$\begin{aligned} \text{(P2)} \quad \min_{(\mathbf{a}, \{\tau^l_i\}, T_{max})} \quad & \beta_e E + \beta_t T_{max}, \\ \text{s.t.} \quad & T_{max} \geq T_1, T_{max} \geq T_2, \\ & \ldots, T_{max} \geq T_O, \\ & 0 \leq \frac{L_i}{\tau^l_i} \leq f_{peak}, \\ & a_i \in \{0, 1\}, \forall i \in \mathcal{M}. \end{aligned} \tag{17}$$

Notice that (P2) is non-convex in general due to the binary variables $\mathbf{a}$. However, for any given $\mathbf{a}$, the remaining optimization over $\{\tau^l_i\}$ is a convex problem. In the following, we assume a fixed offloading decision $\mathbf{a}$ and derive an efficient algorithm to obtain the optimal $(\tau^l_i)^*$, or equivalently the optimal local CPU frequencies $(f^l_i)^*$.

B. Optimal Local CPU Frequencies

Suppose that $\mathbf{a}$ is given. We express a partial Lagrangian of Problem (P2) as

$$\begin{aligned} L(\{\tau^l_i\}, T_{max}, \lambda_1, \ldots, \lambda_O) = & \beta_e E + \beta_t T_{max} \\ & + \sum_{o=1}^{O} \lambda_o (T_o - T_{max}), \end{aligned} \tag{18}$$

where $\{\lambda_o \geq 0, o \in \mathcal{O}\}$ denotes the dual variables associated with the corresponding constraints. Let $\{\lambda^*_o, o \in \mathcal{O}\}$ denote the optimal dual variables. Then, we derive the closed-form expressions for the optimal local CPU frequencies as follows.

Proposition 3.1: $\forall i$ with $a_i = 0$, by denoting the index set of the paths that contain task i as $\Upsilon(i)$, the optimal CPU frequencies at the MD satisfy

$$(f^l_i)^* = \min\left\{ \sqrt[3]{\frac{\sum_{o \in \Upsilon(i)} \lambda^*_o}{2\kappa\beta_e}}, f_{peak} \right\}. \tag{19}$$

Proof: Please refer to Appendix B. ∎

From Proposition 3.1, we observe that the optimal $(f^l_i)^*$ is determined by the dual variables λ^*_o corresponding to all the paths containing task i. Besides, increasing β_e leads to a lower optimal $(f^l_i)^*$ for energy saving.

Corollary 3.1: The summation of the optimal dual variables over all paths is equal to the constant β_t. That is,

$$\sum_{o \in \mathcal{O}} \lambda^*_o = \beta_t. \tag{20}$$

Then, if $\Upsilon(i) = \mathcal{O}$, according to the Proposition 3.1, the optimal local CPU frequency for task i is

$$(f^l_i)^* = \min\left\{ \sqrt[3]{\frac{\beta_t}{2\kappa\beta_e}}, f_{peak} \right\}, \tag{21}$$

which is a constant regardless of the values of $\lambda^*_o, o \in \mathcal{O}$.

Proof: Please refer to Appendix C. ∎

The above corollary indicates that the optimal $(f^l_i)^*$ is a constant when the i-th task is included in all the paths, i.e., $\Upsilon(i) = \mathcal{O}$.

Based on Proposition 3.1 and Corollary 3.1, we can apply the projected subgradient method [27] to search for the optimal

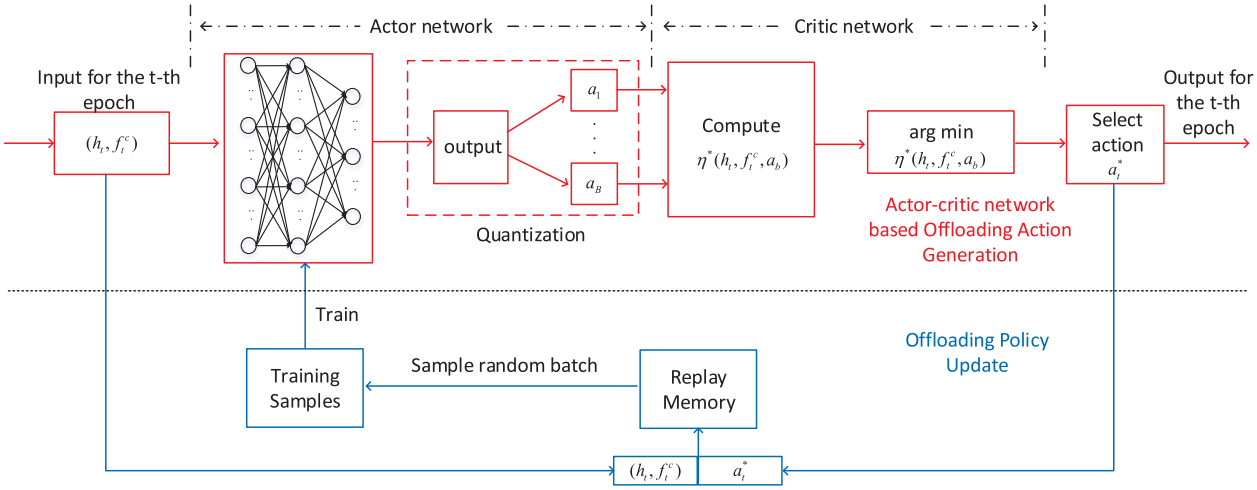


Fig. 3. The schematics of the deep reinforcement learning framework.

dual variables $\{\lambda_o^*, o \in \mathcal{O}\}$. Specifically, we initialize $\{\lambda_o^{(0)} \geq 0, o \in \mathcal{O}\}$ satisfying (20). In the ψ-th iteration, we first calculate $T_o, \forall o \in \mathcal{O}$, using (16) and (19) and set $T_{max} = \max\{T_1, \ldots, T_O\}$. Then, the dual variables are updated to $\{\hat{\lambda}_o^{(\psi)}, o \in \mathcal{O}\}$ by using subgradients $(T_o - T_{max}), \forall o \in \mathcal{O}$, i.e.,

$$\hat{\lambda}_o^{(\psi)} = \lambda_o^{(\psi-1)} - \epsilon(T_o - T_{max}), \tag{22}$$

where ϵ is a small learning rate. In order to guarantee the feasibility of dual variables, we need to project $\{\hat{\lambda}_o^{(\psi)}, o \in \mathcal{O}\}$ to the feasible region given in (20). The projection is calculated from the following convex problem,

$$\begin{aligned} \min_{\{\lambda_o^{(\psi)}\}} \quad & \sqrt{\sum_{o\in\mathcal{O}} (\lambda_o^{(\psi)} - \hat{\lambda}_o^{(\psi)})^2}, \\ \text{s.t.} \quad & \sum_{o\in\mathcal{O}} \lambda_o^{(\psi)} = \beta_t, \\ & \lambda_o^{(\psi)} \geq 0, \forall o \in \mathcal{O}, \end{aligned} \tag{23}$$

which can be efficiently solved by general convex optimization techniques, e.g., interior point method [27]. After updating the dual variables, we can further obtain the updated optimal local CPU frequencies. Such iteration proceeds until a stopping criterion is met. The pseudo-code of the method is shown in Algorithm 1.

IV. Deep Reinforcement Learning Based Task Offloading

In the last section, we efficiently obtain the optimal $\mathbf{f}$ given the offloading decision $\mathbf{a}$. Intuitively, we can enumerate all 2^M feasible $\mathbf{a}$ and choose the optimal one that achieves the minimum objective of (P2). However, such brute-force search is computationally prohibitive, especially when the problem needs to be frequently re-solved with time-varying channel gains and available server computing power. Besides, other searching based methods, such as branch-and-bound and Gibbs sampling algorithms, are also time consuming when M is large.

Algorithm 1 Optimal algorithm for (P2) under fixed offloading decision

1: **initialize** $\{\lambda_o^{(0)} \geq 0\}$ satisfying (20) and set $\psi = 0$.
2: **repeat**
3: Compute $T_o, \forall o \in \mathcal{O}$, using (16) and (19) with given $\{\lambda_o^{(\psi)}\}$.
4: Set $T_{max} = \max\{T_1, \ldots, T_O\}$.
5: Update $\{\lambda_o^{(\psi)}\}$ to $\{\hat{\lambda}_o^{(\psi+1)}\}$ using (22).
6: Project $\{\hat{\lambda}_o^{(\psi+1)}\}$ to the feasible region by solving Problem (23) and set $\psi = \psi + 1$.
7: **until** $\{\lambda_o\}$ converge to a prescribed accuracy.
8: Obtain $\{(f_i^l)^*\}$ by (19).

In this section, we propose a DRL-based algorithm to solve the joint optimization under time-varying channel gains and CPU frequency at the edge server. Our goal is to derive an offloading decision policy π that can quickly predict an optimal offloading action $\mathbf{a}^* \in \{0,1\}^M$ of (P2) once the channel gain $\mathbf{h} = \{h_{k,i}^u, h_{k,i}^d, (k,i) \in \mathcal{E}\}$ and the CPU frequency f^c at the edge server are revealed at the beginning of the execution of the application (task graph). The offloading decision policy is denoted as

$$\pi : \{\mathbf{h}, f^c\} \mapsto \mathbf{a}^*. \tag{24}$$

The algorithm structure is illustrated in Fig. 3. There are two stages in the DRL-based offloading algorithm: one is referred to as the actor-critic network based offloading action generation, and the other is offloading policy update, which are detailed as follows. Furthermore, we propose the one-climb policy to speed up the learning process.

A. Actor-Critic Network Based Offloading Action Generation

1) Actor Network: The offloading action is generated based on a DNN. We denote the embedded parameters of the DNN at the t-th epoch as $\theta_t, t = 1, 2, \ldots$, where θ_1 is randomly initialized following a zero-mean normal distribution. At the t-th epoch, we take the channel gain $\mathbf{h}_t$ and edge CPU frequency

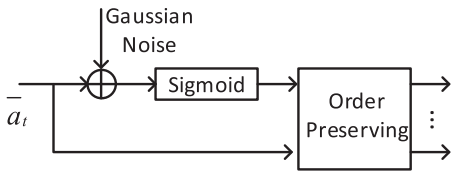


Fig. 4. GNOP quantization method.

f_t^c as the input of the DNN. Accordingly, the DNN outputs a relaxed offloading action $\bar{\mathbf{a}}_t$, which is denoted by a mapping g_{θ_t}, i.e.,

$$\bar{\mathbf{a}}_t = g_{\theta_t}(\mathbf{h}_t, f_t^c), \tag{25}$$

where $\bar{\mathbf{a}}_t = \{\bar{a}_{t,i} \in [0,1], i = 1, \ldots, M\}$, and the $\bar{a}_{t,i}$ denotes the i-th entry of $\bar{\mathbf{a}}_t$.

Notice that each entry of $\bar{\mathbf{a}}_t$ is a continuous value between 0 and 1. To generate a feasible binary offloading decision, we first quantize $\bar{\mathbf{a}}_t$ into B candidate binary offloading actions. Then, the critic network will evaluate the performance of the B candidate actions, and the one with the lowest ETC will be selected as the output solution. Noticeably, for a good quantization method, we only need to generate few candidate actions to reduce the computational complexity. Meanwhile, the quantized actions based on the relaxed action should contain sufficient diversity to yield a lower ETC. In this paper, we propose a Gaussian noise-added order-preserving (GNOP) quantization method as shown in Fig. 4. We define the quantization function as

$$G_B : \bar{\mathbf{a}} \mapsto \Omega_t = \{\mathbf{a}_b | \mathbf{a}_b \in \{0,1\}^M, b = 1, \ldots, B\}, \tag{26}$$

where Ω_t is the generated candidate action set in the t-th epoch.

Order-preserving quantization method was originally introduced to explore the output of the DNN in [21]. The key idea is to preserve the ordering of all the entries in a vector before and after quantization. In our proposed GNOP method, the first $B/2$ actions are generated by traditional order-preserving method, where we assume that B is an even number without loss of generality. Specifically, suppose that the output offloading action is $\bar{\mathbf{a}}_t$. The generation rule for $\{\mathbf{a}_b, b = 1, \ldots, B/2\}$ in the order-preserving method is shown as follow.

First, we obtain the offloading decision $\mathbf{a}_1$ as

$$a_{1,i} = \begin{cases} 1, & \bar{a}_{t,i} > 0.5, \\ 0, & \bar{a}_{t,i} \leq 0.5, \end{cases} \tag{27}$$

for $i = 1, \ldots, M$. For the other $B/2 - 1$ offloading actions, we first order the entries of $\bar{\mathbf{a}}_t$ according to their distances to 0.5, i.e., $|\bar{a}_{t,(1)} - 0.5| \leq |\bar{a}_{t,(2)} - 0.5| \leq \ldots \leq |\bar{a}_{t,(i)} - 0.5| \leq \ldots \leq |\bar{a}_{t,(M)} - 0.5|$, where $\bar{a}_{t,(i)}$ is denoted as the i-th order entry of $\bar{\mathbf{a}}_t$. Then, the b-th offloading action $\mathbf{a}_b$ is obtained as

$$a_{b,i} = \begin{cases} 1, & \bar{a}_{t,i} > \bar{a}_{t,(b-1)}, \\ 1, & \bar{a}_{t,i} = \bar{a}_{t,(b-1)} \text{ and } \bar{a}_{t,(b-1)} < 0.5, \\ 0, & \bar{a}_{t,i} = \bar{a}_{t,(b-1)} \text{ and } \bar{a}_{t,(b-1)} > 0.5, \\ 0, & \bar{a}_{t,i} < \bar{a}_{t,(b-1)}, \end{cases} \tag{28}$$

for $i = 1, \ldots, M$ and $b = 2, \ldots, B/2$.

Compared to the traditional K-nearest neighbor (KNN) method, the order-preserving quantization method leads to a higher diversity in the offloading action space. However, the offloading actions produced by conventional order-preserving quantization method are still closely placed around $\bar{\mathbf{a}}_t$, which reduces the chance of finding a local optimum in a large action space. To better explore the action space, we introduce a Gaussian noise-added approach to generate the other half of $B/2$ candidate actions. Specifically, we first add a Gaussian noise to $\bar{\mathbf{a}}_t$ as

$$\ddot{\mathbf{a}}_t = f_{sg}(\bar{\mathbf{a}}_t + \mathbf{n}), \tag{29}$$

where $\mathbf{n} \sim \mathcal{N}(0,1)$ and $f_{sg}(\cdot)$ is the sigmoid function that maps the original noise-added action to $\ddot{\mathbf{a}}_t \in [0,1]$. Then, we apply the order-preserving method on $\ddot{\mathbf{a}}_t$ to generate the $B/2$ offloading actions.

2) Critic Network: After generating the candidate offloading actions in the actor network, we evaluate the ETC performance of each action in the critic network. Instead of training a critic DNN as the conventional actor-critic method does, we can accurately and efficiently evaluate the ETC corresponding to each candidate $\mathbf{a}_b$ using our analysis in Section III. In particular, we denote the ETC achieved by the candidate $\mathbf{a}_b$ as $\eta^*(\mathbf{h}_t, f_t^c, \mathbf{a}_b)$ by optimizing the local CPU frequencies $\mathbf{f}$ as described in Algorithm 1. This greatly reduces the training cost of the critic DNN and increases the accuracy of ETC evaluation. Accordingly, we choose the best offloading action $\mathbf{a}_t^*$ at the t-th epoch as

$$\mathbf{a}_t^* = \arg\min_{\mathbf{a}_b \in \Omega_t} \eta^*(\mathbf{h}_t, f_t^c, \mathbf{a}_b). \tag{30}$$

Noticeably, $\mathbf{a}_t^*$, together with its corresponding optimal resource allocation $\mathbf{f}^*$ constitutes the optimal solution to Problem (P1) (or equivalently, Problem (P2)).

B. Offloading Policy Update

The optimal actions learned in the offloading action generation stage are used to update the parameters of the DNN through the offloading policy update stage.

As illustrated in Fig. 3, we implement a replay memory to store the past state-action pairs, where the memory is of limited capacity. At the t-th epoch, $(\{\mathbf{h}_t, f_t^c\}, \mathbf{a}_t^*)$ obtained in the actor-critic network based offloading action generation stage is added to the memory as a new training data sample. Note that the newly generated data sample will replace the oldest one if the memory is full.

The data samples stored in the memory are used to train the DNN. Specifically, in the t-th epoch, we randomly select a batch of training data samples $\{(\{\mathbf{h}_\omega, f_\omega^c\}, \mathbf{a}_\omega^*), \omega \in \mathcal{T}_t\}$ from the memory, where $\mathcal{T}_t$ represents the set of chosen time indices. Then, we minimize the average cross-entropy loss $Loss(\theta_t)$ through the Adam algorithm in order to update the parameters θ_t of the DNN, where

$$Loss(\theta_t) = -\frac{1}{|\mathcal{T}_t|} \sum_{\omega \in \mathcal{T}_t} \Big((\mathbf{a}_\omega^*)^\top \log g_{\theta_t}(\mathbf{h}_\omega, f_\omega^c) + (1 - \mathbf{a}_\omega^*)^\top \log(1 - g_{\theta_t}(\mathbf{h}_\omega, f_\omega^c)) \Big). \tag{31}$$

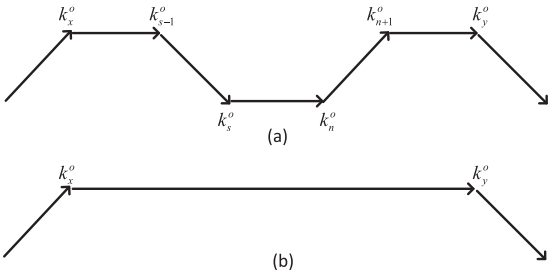


Fig. 5. Illustration of a two-time offloading and an one-climb schemes in a path o.

TABLE I
SIMULATION PARAMETERS

$W = 2 \times 10^6$ Hz	$\kappa = 10^{-26}$
$\sigma^2 = 10^{-10}$ Watt	$f_{peak} = 0.01$ GHz
$P_{MD} = 0.1$ Watt	$f^c \sim \mathcal{U}(2, 50)$ GHz
$P_{AP} = 1$ Watt	$d = 20$ meters
$A_d = 4.11$	$f_c = 915$ MHz
$PL = 3$	$\beta_t = 0.5$
$\beta_e = 0.5$	

$|\mathcal{T}_t|$ is the size of $\mathcal{T}_t$, the superscript $\top$ denotes the transpose operator, and the log function is the element-wise logarithm operation for a vector. For brevity, the detail of the Adam algorithm is omitted here. In practice, we start the training step when the number of samples is larger than half of the memory size and train the DNN in every δ epochs in order to collect a sufficient number of new data samples in the memory.

C. Low-Complexity Action Generation Method

Within the proposed DRL framework, we improve the GNOP quantization method to further reduce the complexity. The basic idea is to restrict our action selection only to those offloading decisions that satisfy the following one-climb policy.

Definition 3 (One-climb policy): The execution for the tasks in each path of the graph G migrates at most once from the MD to the edge server.

Fig. 5 illustrates the two-time offloading and one-climb schemes in a path o. We show in the Appendix D that by converting the scheme from the two-time offloading to the one-climb policy, the MD saves the energy and time costs for the path o. This however may increase the ETC of other paths with overlapping tasks with path o. We show that, certain mild conditions hold if the minimum ETC is achieved when all the paths satisfy the one-climb policy. Please refer to Appendix D for the detailed analysis.

The one-climb policy is applied to reduce the number of offloading actions to be evaluated by the critic network. Suppose that $\Omega_t = \{\mathbf{a}_b | \mathbf{a}_b \in \{0,1\}^M, b = 1, \ldots, B\}$ is the set of actions obtained by the GNOP quantization method at the t-th epoch. We remove the actions in Ω_t that violate the one-climb policy. By using the one-climb policy in the quantization module, we efficiently reduce the number of calculations for Algorithm 1 at the actor-critic network based offloading action generation stage.

V. NUMERICAL RESULTS

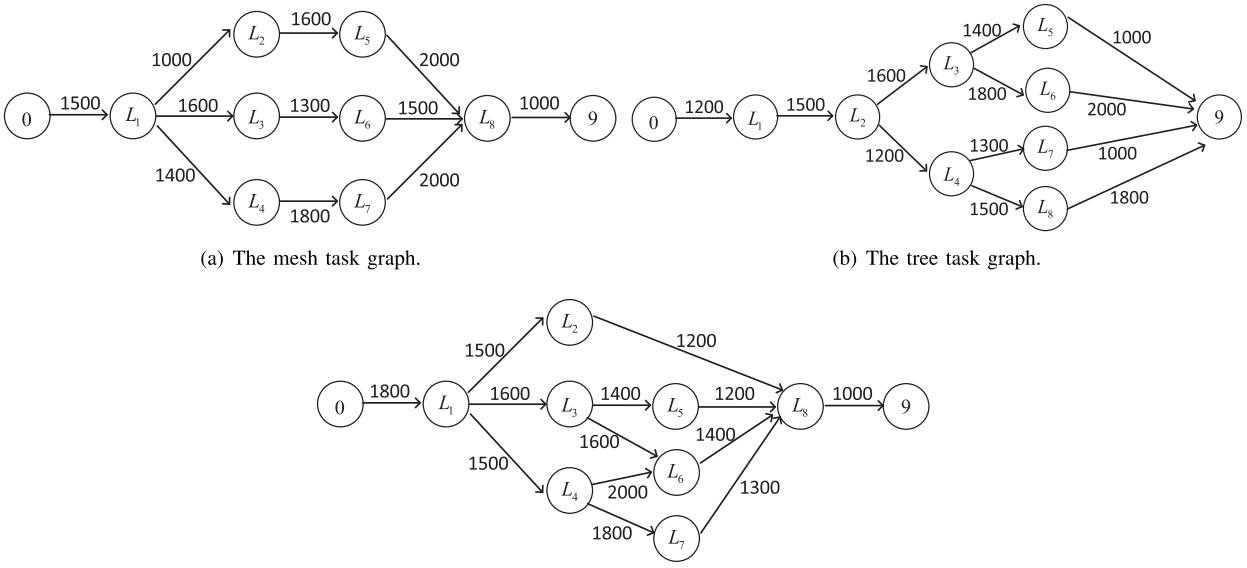
In this section, we evaluate the performance of our proposed algorithm through numerical simulations. Consider three different task graphs in Fig. 6, each consisting of 8 actual tasks. Fig. 6(a) illustrates a mesh task graph including a set of linear chains, while a task graph with tree-based structure is considered in Fig. 6(b). In Fig. 6(c), we consider a general task graph which is a combination of the mesh and the tree. The input and output data size (KByte) of each task are shown in Fig. 6. We assume that the computing workload $\{L_i\} = [60.5\ 80.3\ 152.6\ 105.8\ 195.3\ 86.4\ 166.8\ 100.3]$ (Mcycles) for all the three task graphs. The transmit power at the MD and the AP are fixed as 100 mW and 1 W, respectively. It is assumed that the CPU frequency f^c is time-varying and follows a uniform distribution between 2 GHz and 50 GHz. Besides, the peak computational frequency of the MD is equal to 0.01 GHz.

In the simulations, we assume that the average channel gain $\bar{h}_{k,i}$ follows the free-space path loss model $\bar{h}_{k,i} = A_d(\frac{3\cdot 10^8}{4\pi f_c d})^{PL}$, where $A_d = 4.11$ denotes the antenna gain, $f_c = 915$ MHz denotes the carrier frequency, $d = 20$ in meters denotes the distance between the MD and the AP, and $PL = 3$ denotes the pass loss exponent. The time-varying fading channel $h_{k,i}^u$ follows an i.i.d. Rician distribution, where the LOS link power is equal to $0.6\bar{h}_{k,i}$. Besides, we follow some classic uplink-downlink channel models that the random variable downlink channel $h_{k,i}^d$ is correlated with the uplink channel $h_{k,i}^u$ and we set the correlation coefficient as 0.7 (the coefficient 0.7 is used in [28] for modeling weakly-correlated uplink and downlink channels. For some highly correlated case, the correlation coefficient is larger than 0.9). The noise power $\sigma^2 = 10^{-10}$ W. In addition, we set the computing efficiency parameter $\kappa = 10^{-26}$, and the bandwidth $W = 2$ MHz. The priority weights of energy consumption and computation time of the MD are set as $\beta_t = \beta_e = 0.5$. The parameters used in the simulations are listed in Table I.

We consider a fully connected DNN consisting of one input layer, three hidden layers, and one output layer in the proposed DRL algorithm, where the first, second, and third hidden layers have 160, 120, and 80 hidden neurons, respectively. We implement the DRL algorithm in Python with TensorFlow and set the learning rate for Adam optimizer as 0.01, the training batch size $|\mathcal{T}| = 128$, the memory size as 1024, and the training interval $\delta = 10$.

A. Convergence Performance

Without loss of generality, we first consider the tree task graph in Fig. 6(b) as an example to study the impact of the parameters on the convergence performance of the proposed DRL algorithm, including learning rates, batch sizes, memory sizes, and learning intervals in Fig. 7. As shown in Fig. 7(a), we illustrate the impact of the learning rate in Adam optimizer on the moving average of the training loss over moving windows of 15 epochs. It is observed that a too large (i.e., 0.1) or a too small (i.e., 0.001) learning rate leads to a worse convergence. Therefore, in the following simulations, we set the learning rate as 0.01. As for different batch sizes

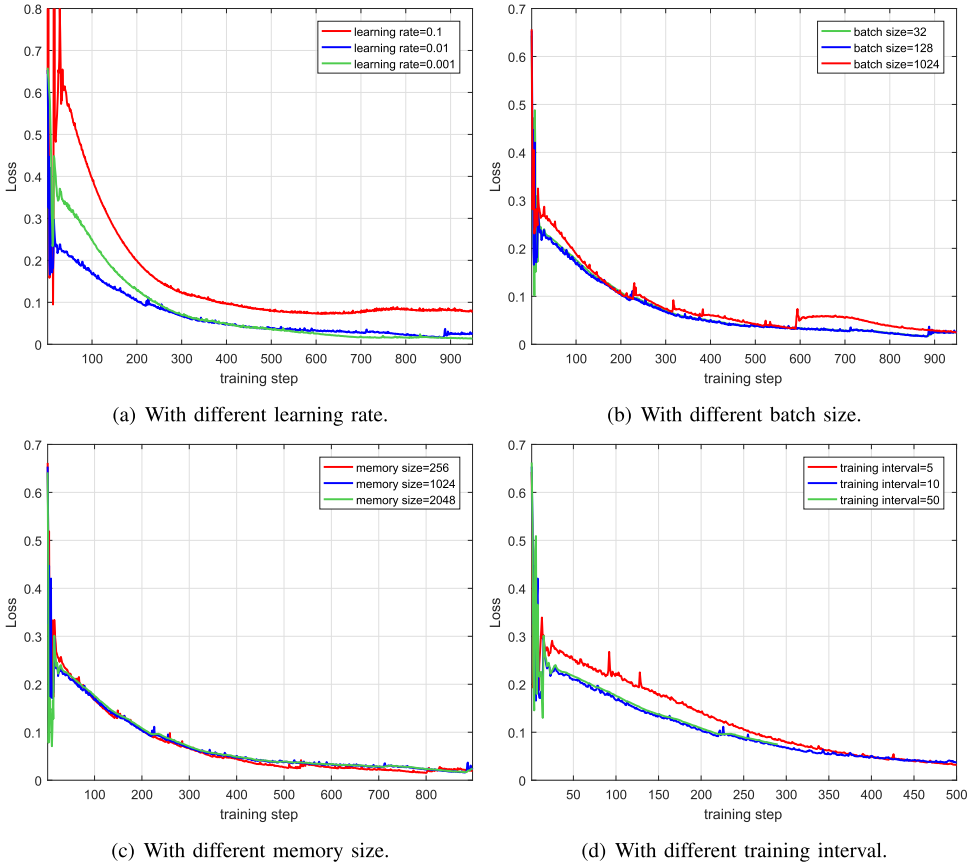


(a) The mesh task graph.

(b) The tree task graph.

(c) The general task graph.

Fig. 6. The considered task graphs in the simulation.



(a) With different learning rate.

(b) With different batch size.

(c) With different memory size.

(d) With different training interval.

Fig. 7. Moving average of the training loss for the tree task graph with different parameters.

in Fig. 7(b), we observe that a large batch size (i.e., 1024) causes higher fluctuation for the moving average of the training loss, which is due to the frequent usage of the "old" training data in the memory. Besides, a large batch size consumes more time when training the DNN. Hence, the training batch size is set to 128 in the following simulations. In Fig. 7(c), the moving average of the training loss gradually decreases and stabilizes at around 0.01 for different memory sizes. In addition, we observe that the convergence performance is insensitive to the memory size. In Fig. 7(d), we investigate the

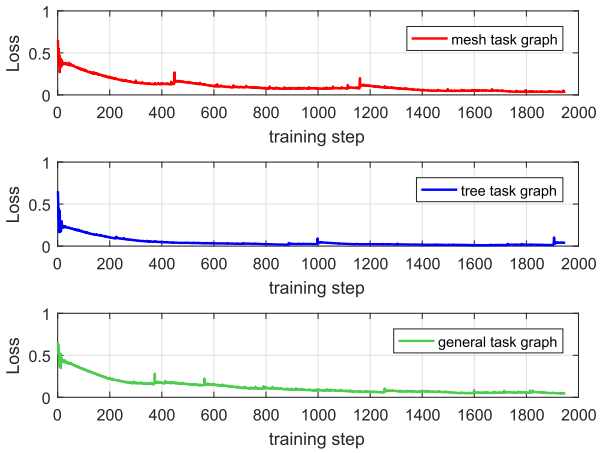


Fig. 8. Moving average of the training loss for the three task graphs when the learning rate is 0.01, the training batch size is 128, the memory size is 1024, and the training interval is 10.

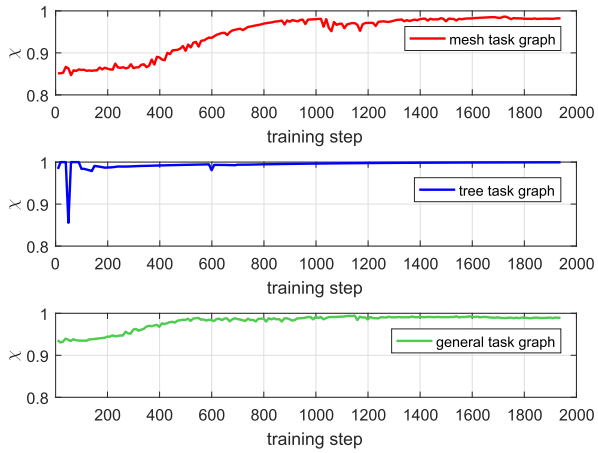


Fig. 9. Moving average of the accuracy rates over training steps for the three task graphs when the learning rate is 0.01, the training batch size is 128, the memory size is 1024, and the training interval is 10.

convergence of our proposed DRL algorithm under different training intervals. It is observed that for different training intervals, the moving average of the training loss gradually decreases and becomes stable at around 0.02 after 400 training steps, which means that the convergence performance is insensible with respect to the training intervals. In the following simulations, we set the training interval as 10.

Accordingly, Fig. 8 illustrates the convergence performance of the DRL algorithm for the three task graphs, where we set the learning rate as 0.01, the training batch size as 128, the memory size as 1024, and the training interval as 10. We observe that under different task graphs, the moving average of the training loss is below 0.1 after 300 training steps.

In Fig. 9, we plot the moving average of the accuracy rates over training steps for the three task graphs, where the proposed DRL algorithm is tested in each training step using 50 independent realizations. We define the accuracy rate as $\chi = 1 - \frac{\eta_{DRL} - \eta^*}{\eta^*}$, where η^* is the average optimal ETC obtained by the exhaustive search method under the 50 independent realizations and $\frac{\eta_{DRL} - \eta^*}{\eta^*}$ is the ratio of bias of the ETC in DRL algorithm compared to the optimum. We see that the moving average of the accuracy rates for the proposed DRL algorithm gradually converges as the training step increases. Specifically, for the mesh task graph, the achieved χ exceeds 0.99 after 800 training steps.

B. Energy and Time Cost (ETC) Performance Evaluation

We now compare the energy and time cost (ETC) performance of the proposed methods with that of the following four representative benchmarks.

- Gibbs sampling algorithm. The Gibbs sampling algorithm updates the offloading decision iteratively based on the designed probability distribution with respect to the objective values and the temperature parameter. According to the proof in [29], a Gibbs sampling algorithm obtains the optimal solution when it converges.

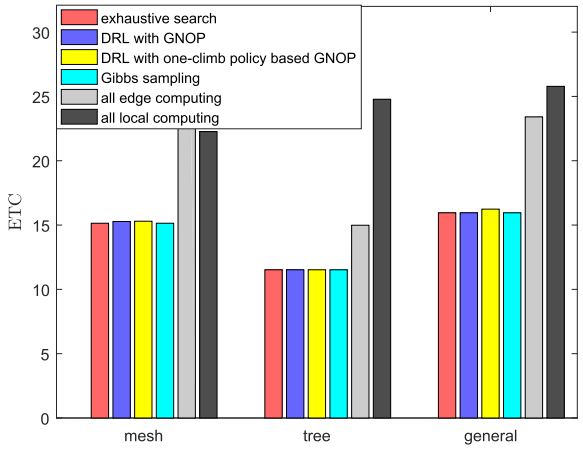


Fig. 10. Comparisons of ETC performance for different offloading algorithms.

- Exhaustive search. We enumerate all 2^M feasible offloading decisions and choose the optimal one that yields the minimum ETC.
- All edge computing. In this scheme, all the tasks of the MD are offloaded to the edge side for execution.
- All local computing. In this scheme, all the tasks of the MD are executed locally.

In Fig. 10, we compare the ETC performance among different offloading schemes under the three task topologies in Fig. 6. Each point in the figure is the average performance of 50 independent realizations. When evaluating the performance, we have neglected the first 20000 time epochs as a warm-up period, so that the DRL has converged. We observe that for all the three task graphs, our proposed DRL algorithm can achieve near-optimal performance compared with the exhaustive search and the Gibbs sampling algorithms. In addition, by applying the one-climb policy heuristics in the GNOP quantization method, the ETC performance is hardly affected. Besides, the DRL algorithm significantly outperforms the all-edge-computing and all-local-computing schemes. This

TABLE II
ACCURACY RATES χ FOR DIFFERENT TASK GRAPHS

	Mesh	Tree	General
χ	99.1%	99.9%	99.9%

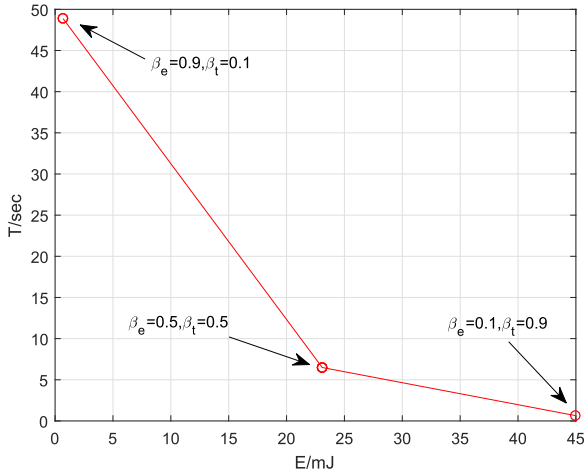


Fig. 11. The tradeoff between the total execution time and energy consumption of the MD under different weights for the tree task graph.

suggests the benefit of adapting the offloading decisions under different wireless channels and edge CPU frequency.

Then, Table II illustrates the average accuracy rates of our proposed DRL algorithm. It is observed that on average the DRL algorithm achieves over 99.1% of the optimal ETC. Specifically, for the general task graph shown in Fig. 6(c), 99.9% accuracy rate with respect to the ETC objective is achieved.

In Fig. 11, we illustrate the tradeoff between the total execution time and energy consumption of the MD under different weights for the tree task graph. By setting $\beta_e + \beta_t = 1$, we observe that with the increase of β_e, the MD achieves lower energy consumption but higher execution time. Notice that the values of the energy consumption E and the total execution time T are within a similar range (from 0 to about 50). In fact, even if their values are within rather dissimilar ranges (e.g., 0-1000 for E and 0-5 for T), we may simply scale the values of E or T using proper weighting parameters to ensure that the weighted sum performance can reflect non-trivial performance tradeoff, i.e., one objective does not dominate the other when optimizing the offloading decisions and resource allocation. In Fig. 12, we further compare the ETC performance for different offloading algorithms under different weights in the tree task graph. When β_e and β_t vary, our proposed DRL algorithm is also applicable and we can obtain the same insights as observed in Fig. 10. We find that the proposed DRL algorithm achieves close-to-optimal performance under different β_e and β_t. Therefore, without loss of generality, we consider $\beta_e = \beta_t = 0.5$ in the rest of simulations.

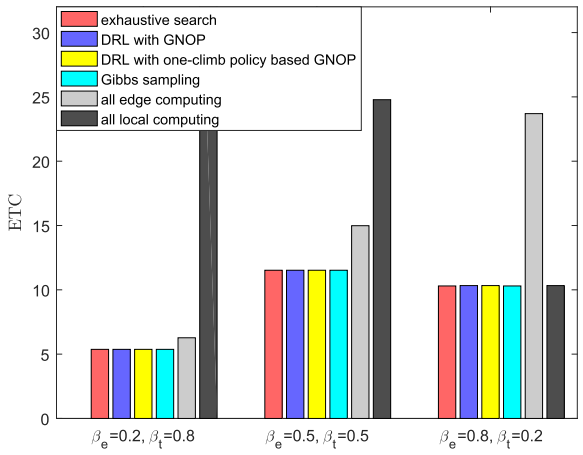


Fig. 12. Comparisons of ETC performance for different offloading algorithms under different weights in the tree task graph.

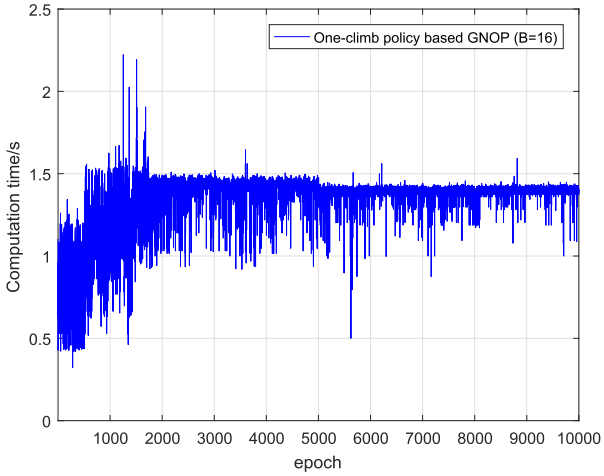


Fig. 13. Computation time for each epoch under the tree task graph.

C. Complexity of the Proposed DRL Algorithm

At last, we compare the computational complexity among the four algorithms, where the number of quantized offloading decisions for each epoch in the DRL algorithm $B = 16$. We see from the Table III that the DRL algorithm with one-climb policy based GNOP quantization significantly reduces the computation time compared with the DRL algorithm with GNOP method. That is, around 37.15%, 4.86%, and 33.86% lower average runtime achieved in the mesh, tree, and general task graphs, respectively. Therefore, the one-climb policy heuristics can achieve the near performance as the original GNOP method, while efficiently reducing the complexity of the proposed DRL algorithm. Specifically, in Fig. 13, we illustrate the computation time for each epoch in the DRL algorithm with one-climb policy based GNOP method under the tree task graph. For some epochs, the DRL algorithm with one-climb policy based GNOP only consumes around 0.3 second for obtaining the optimal solution.

Furthermore, as shown in Table III, the DRL algorithm with one-climb policy based GNOP requires much shorter runtime than the Gibbs sampling algorithm and the exhaustive search

TABLE III
COMPARISIONS OF AVERAGE COMPUTATION TIME FOR EACH REALIZATION

	Mesh	Tree	General
DRL with One-climb policy based GNOP ($B = 16$)	**0.9240 s**	**1.3421 s**	**1.0464 s**
DRL with GNOP ($B = 16$)	1.4702 s	1.4107 s	1.5821 s
Gibbs sampling	8.2039 s	8.3046 s	8.6101 s
Exhaustive search	25.6690 s	26.8181 s	27.5185 s

method. In particular, for the general task graph, it outputs an offloading decision in around 1 second for each realization on average, while the Gibbs sampling and exhaustive search methods spend 8 times and 26 times longer runtime, respectively.

VI. CONCLUSION

Considering a single-user MEC system with a general task graph, this paper has proposed a DRL framework to jointly optimize the offloading decisions and resource allocation, with the goal of minimizing the weighted sum of MD's energy consumption and task execution time. The DRL framework utilizes a DNN to learn and improve the offloading policy from the experiences, which completely removes the need of solving hard combinatorial optimization problem. Besides, we have derived a Gaussian noise-added order-preserving quantization method to efficiently generate offloading actions in the DRL framework. Meanwhile, a low-complexity algorithm has been proposed to accurately evaluate the ETC performance of each generated offloading decision. We have further proposed an one-climb policy to speed up the learning process. Simulation results have demonstrated that the proposed algorithm can achieve near-optimal performance while significantly decreasing the complexity compared to the conventional optimization methods.

APPENDIX A
PROOF OF LEMMA 3.1

According to (5), (6), (11) and (12), we have

$$\begin{aligned} FT^l_{M+1} &= RT^l_{M+1} + \tau^l_{M+1} \\ &= \max_{k_m \in \mathbf{pred}(M+1)} \Big\{ (1-a_{k_m}) FT^l_{k_m} \\ &\quad + a_{k_m} (FT^c_{k_m} + \tau^d_{k_m,M+1}) \Big\} \\ &= \max_{k_m \in \mathbf{pred}(M+1)} \Big\{ (1-a_{k_m}) (RT^l_{k_m} + \tau^l_{k_m}) \\ &\quad + a_{k_m} (RT^c_{k_m} + \tau^c_{k_m} + \tau^d_{k_m,M+1}) \Big\}. \end{aligned} \tag{32}$$

For the term $RT^l_{k_m}$ in (32), we have

$$\begin{aligned} RT^l_{k_m} &= \max_{k_{m-1} \in \mathbf{pred}(k_m)} \Big\{ (1-a_{k_{m-1}}) FT^l_{k_{m-1}} \\ &\quad + a_{k_{m-1}} (FT^c_{k_{m-1}} + \tau^d_{k_{m-1},k_m}) \Big\} \\ &= \max_{k_{m-1} \in \mathbf{pred}(k_m)} \Big\{ (1-a_{k_{m-1}}) (RT^l_{k_{m-1}} + \tau^l_{k_{m-1}}) \\ &\quad + a_{k_{m-1}} (RT^c_{k_{m-1}} + \tau^c_{k_{m-1}} + \tau^d_{k_{m-1},k_m}) \Big\}. \end{aligned} \tag{33}$$

For the term $RT^c_{k_m}$ in (32), we have

$$\begin{aligned} RT^c_{k_m} &= \max_{k_{m-1} \in \mathbf{pred}(k_m)} \Big\{ (1-a_{k_{m-1}}) (FT^l_{k_{m-1}} + \tau^u_{k_{m-1},k_m}) \\ &\quad + a_{k_{m-1}} FT^c_{k_{m-1}} \Big\} \\ &= \max_{k_{m-1} \in \mathbf{pred}(k_m)} \Big\{ (1-a_{k_{m-1}}) (RT^l_{k_{m-1}} + \tau^l_{k_{m-1}} \\ &\quad + \tau^u_{k_{m-1},k_m}) + a_{k_{m-1}} (RT^c_{k_{m-1}} + \tau^c_{k_{m-1}}) \Big\}. \end{aligned} \tag{34}$$

Substituting (33) and (34) into (32), we have FT^l_{M+1} as in (35), shown at the bottom of the next page, where T_o is defined in (16).

APPENDIX B
PROOF OF PROPOSITION 3.1

The derivative of L of (18) with respect to τ^l_i can be expressed as

$$\frac{\partial L}{\partial \tau^l_i} = -\frac{2\kappa\beta_e (L_i)^3}{(\tau^l_i)^3} + \sum_{o \in \Upsilon(i)} \lambda_o, \tag{36}$$

where $\frac{\partial L}{\partial \tau^l_i}$ is a monotonously increasing function with $\tau^l_i \in [\frac{L_i}{f_{peak}}, +\infty)$. Thus, if $\frac{\partial L}{\partial \tau^l_i}|_{\tau^l_i = \frac{L_i}{f_{peak}}} > 0$, we have $(f^l_i)^* = f_{peak}$. Otherwise, we have

$$\tau^l_i = L_i \sqrt[3]{\frac{2\kappa\beta_e}{\sum_{o \in \Upsilon(i)} \lambda_o}} \Rightarrow (f^l_i)^* = \frac{L_i}{\tau^l_i} = \sqrt[3]{\frac{\sum_{o \in \Upsilon(i)} \lambda^*_o}{2\kappa\beta_e}}. \tag{37}$$

Hence,

$$(f^l_i)^* = \min \left\{ \sqrt[3]{\frac{\sum_{o \in \Upsilon(i)} \lambda^*_o}{2\kappa\beta_e}}, f_{peak} \right\}. \tag{38}$$

APPENDIX C
PROOF OF COROLLARY 3.1

The derivative of L of (18) with respect to T_{max} can be expressed as

$$\frac{\partial L}{\partial T_{max}} = \beta_t - \sum_{o=1}^{O} \lambda_o. \tag{39}$$

By setting $\frac{\partial L}{\partial T_{max}} = 0$, we have

$$\sum_{o=1}^{O} \lambda_o^* = \beta_t. \tag{40}$$

APPENDIX D
OPTIMALITY ANALYSIS FOR ONE-CLIMB POLICY

In the following, we analyze the optimality of the one-climb policy. Suppose that there exists a path o in the task graph, where the optimal offloading decision allows the MD to offload its task data for two times. Under the two-time offloading scheme, for the tasks in $\Psi(o) = \{0, k_1^o, \ldots, k_x^o, \ldots, k_{s-1}^o, k_s^o, \ldots, k_n^o, k_{n+1}^o, \ldots, k_y^o, \ldots, M+1\}$, tasks from k_x^o to k_{s-1}^o are migrated to the edge server for execution. Then, tasks from k_s^o to k_n^o prefer local computing, followed by tasks from k_{n+1}^o to k_y^o migrated to the edge server. We also consider an one-climb scheme for performance comparison, where tasks from k_x^o to k_y^o are executed on the edge server.

We denote the optimal offloading decision and local CPU frequencies in the two-time and one-climb offloading schemes as $\{\hat{\mathbf{a}}, \hat{\mathbf{f}}\}$ and $\{\tilde{\mathbf{a}}, \tilde{\mathbf{f}}\}$, respectively. By the optimality assumption, we have $\eta(\hat{\mathbf{a}}, \hat{\mathbf{f}}) < \eta(\tilde{\mathbf{a}}, \tilde{\mathbf{f}})$.

For the two-time offloading policy in path o, the total execution time from the k_x^o-th task to the k_y^o-th task can be expressed as

$$\begin{aligned}\hat{T}_o^{k_x^o \sim k_y^o} = &\sum_{m=x}^{s-1} (\tau_{k_m^o}^c) + \tau_{k_{s-1}^o, k_s^o}^d + \sum_{m=s}^{n} (\tau_{k_m^o}^l) \\ &+ \tau_{k_n^o, k_{n+1}^o}^u + \sum_{m=n+1}^{y} (\tau_{k_m^o}^c).\end{aligned} \tag{41}$$

As for the one-climb policy in path o, we have

$$\tilde{T}_o^{k_x^o \sim k_y^o} = \sum_{m=x}^{y} \tau_{k_m^o}^c. \tag{42}$$

Since $f^c > f_{peak}$, the following inequalities hold for the k_s^o-th and k_n^o-th tasks:

$$\tau_{k_s^o}^c < \tau_{k_s^o}^l < \tau_{k_s^o}^l + \tau_{k_{s-1}^o, k_s^o}^d, \tag{43}$$

$$\tau_{k_n^o}^c < \tau_{k_n^o}^l < \tau_{k_n^o}^l + \tau_{k_n^o, k_{n+1}^o}^u. \tag{44}$$

In addition, we have $\tau_{k_m^o}^c < \tau_{k_m^o}^l, m = s, \ldots, n$ for the tasks in the o-th path between k_s^o and k_n^o. Therefore, it can be shown that $\hat{T}_o^{k_x^o \sim k_y^o} > \tilde{T}_o^{k_x^o \sim k_y^o}$.

On the other hand, with respect to the energy consumption of the MD from the k_x^o-th task to the k_y^o-th task in the o-th path, we observe that the two-time offloading scheme consumes more energy compared with the one-climb policy due to the local tasks computing $e_{k_i^o}^l$ from k_s^o to k_n^o and the k_{n+1}^o-th task's offloading $e_{k_n^o, k_{n+1}^o}^u$. That is, $\hat{E}_o^{k_x^o \sim k_y^o} > \tilde{E}_o^{k_x^o \sim k_y^o}$, where $\hat{E}_o^{k_x^o \sim k_y^o}$ and $\tilde{E}_o^{k_x^o \sim k_y^o}$ denote the energy consumption from the k_x^o-th task to the k_y^o-th task in the o-th path under the two-time and one-climb offloading schemes, respectively.

For another path o' in the task graph G, we assume that in the one-climb scheme, tasks from $k_x^{o'}$ to $k_y^{o'}$ are executed on the edge server. Consider the tasks in $\{k_s^o, \ldots, k_n^o\}$ that the path o' also contains. If $\{k_s^o, \ldots, k_n^o\} \bigcap \Psi(o') = \emptyset$, we have $\tilde{T}_{o'} = \hat{T}_{o'}$, where $\tilde{T}_{o'}$ is the total execution time in the o'-th path under one-climb policy, and $\hat{T}_{o'}$ is the execution time when the tasks in $\{k_s^o, \ldots, k_n^o\} \bigcap \Psi(o')$ choose to perform local computing due to the two-time offloading scheme in the o-th path. Meanwhile, $\tilde{E}_{o'} = \hat{E}_{o'}$, where $\tilde{E}_{o'}$ is the total energy consumption in the o'-th path under one-climb policy, and $\hat{E}_{o'}$ is the energy consumption when the tasks in $\{k_s^o, \ldots, k_n^o\} \bigcap \Psi(o')$ change their offloading decisions due to the two-time offloading scheme in the o-th path. Otherwise, if $\{k_s^o, \ldots, k_n^o\} \bigcap \Psi(o') \neq \emptyset$, we consider the following four cases.

- As shown in Fig. 14(a), suppose that the tasks in $\{k_s^o, \ldots, k_n^o\}$, which the path o' also includes, are the first z tasks offloaded to the edge in path o' under one-climb scheme, i.e., $\{k_s^o, \ldots, k_n^o\} \bigcap \Psi(o') =$

$$\begin{aligned}
FT_{M+1}^l = &\max_{k_m \in \mathbf{pred}(M+1)} \left\{(1-a_{k_m})\tau_{k_m}^l + a_{k_m}(\tau_{k_m}^c + \tau_{k_m, M+1}^d)\right\} + \max_{k_m \in \mathbf{pred}(M+1)} \max_{k_{m-1} \in \mathbf{pred}(k_m)} \\
&\left\{(1-a_{k_{m-1}})\tau_{k_{m-1}}^l + a_{k_{m-1}}\tau_{k_{m-1}}^c + a_{k_m}(1-a_{k_{m-1}})\tau_{k_{m-1}, k_m}^u + (1-a_{k_m})a_{k_{m-1}}\tau_{k_{m-1}, k_m}^d\right\} \\
&+ \max_{k_m \in \mathbf{pred}(M+1)} \max_{k_{m-1} \in \mathbf{pred}(k_m)} \left\{(1-a_{k_{m-1}})RT_{k_{m-1}}^l + a_{k_{m-1}}RT_{k_{m-1}}^c\right\} \\
= &\max_{k_m \in \mathbf{pred}(M+1)} \left\{(1-a_{k_m})\tau_{k_m}^l + a_{k_m}(\tau_{k_m}^c + \tau_{k_m, M+1}^d)\right\} + \max_{k_m \in \mathbf{pred}(M+1)} \max_{k_{m-1} \in \mathbf{pred}(k_m)} \\
&\left\{(1-a_{k_{m-1}})\tau_{k_{m-1}}^l + a_{k_{m-1}}\tau_{k_{m-1}}^c + a_{k_m}(1-a_{k_{m-1}})\tau_{k_{m-1}, k_m}^u + (1-a_{k_m})a_{k_{m-1}}\tau_{k_{m-1}, k_m}^d\right\} \\
&+ \max_{k_m \in \mathbf{pred}(M+1)} \max_{k_{m-1} \in \mathbf{pred}(k_m)} \max_{k_{m-2} \in \mathbf{pred}(k_{m-1})} \left\{(1-a_{k_{m-2}})\tau_{k_{m-2}}^l + a_{k_{m-2}}\tau_{k_{m-2}}^c \right. \\
&\left. + a_{k_{m-1}}(1-a_{k_{m-2}})\tau_{k_{m-2}, k_{m-1}}^u + (1-a_{k_{m-1}})a_{k_{m-2}}\tau_{k_{m-2}, k_{m-1}}^d\right\} + \ldots \\
&+ \max_{k_m \in \mathbf{pred}(M+1)} \max_{k_{m-1} \in \mathbf{pred}(k_m)} \cdots \max_{k_1 \in \mathbf{pred}(k_2)} \max_{0 \in \mathbf{pred}(k_1)} \left\{a_{k_1}\tau_{0, k_1}^u\right\} \\
= &\max\{T_1, T_2, \ldots, T_o, \ldots, T_O\}.
\end{aligned} \tag{35}$$

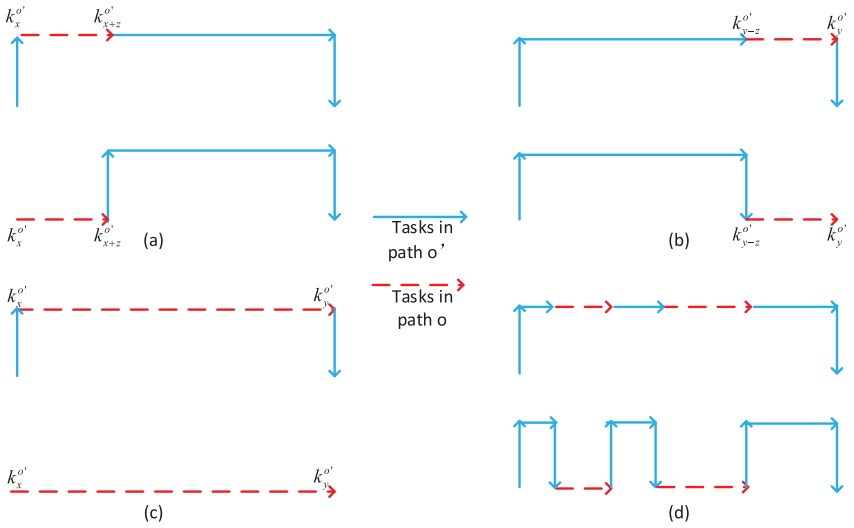


Fig. 14. Illustration of different offloading decisions at the path o' due to the overlapping tasks belonging to path o.

$\{k_x^{o'}, k_{x+1}^{o'}, \ldots, k_{x+z}^{o'}\}$. We have

$$\hat{T}_{o'} - \tilde{T}_{o'} = \frac{O_{k_{x+z}^{o'}, k_{x+z+1}^{o'}}}{R^u(h^u_{k_{x+z}^{o'}, k_{x+z+1}^{o'}})} - \frac{O_{k_{x-1}^{o'}, k_x^{o'}}}{R^u(h^u_{k_{x-1}^{o'}, k_x^{o'}})} + Y - Z, \tag{45}$$

and

$$\hat{E}_{o'} - \tilde{E}_{o'} = P_{MD}\left[\frac{O_{k_{x+z}^{o'}, k_{x+z+1}^{o'}}}{R^u(h^u_{k_{x+z}^{o'}, k_{x+z+1}^{o'}})} - \frac{O_{k_{x-1}^{o'}, k_x^{o'}}}{R^u(h^u_{k_{x-1}^{o'}, k_x^{o'}})}\right] + X, \tag{46}$$

where X, Y, Z are the total local execution energy consumption, local computing time and edge execution time among the tasks $\{k_s^o, \ldots, k_n^o\} \bigcap \Psi(o')$ in the path o', respectively. In this case, if $\hat{T}_{o'} > \tilde{T}_{o'}$ and $\hat{E}_{o'} > \tilde{E}_{o'}$ hold, the following inequality needs to be satisfied:

$$\Delta^u = \frac{O_{k_{x+z}^{o'}, k_{x+z+1}^{o'}}}{R^u(h^u_{k_{x+z}^{o'}, k_{x+z+1}^{o'}})} - \frac{O_{k_{x-1}^{o'}, k_x^{o'}}}{R^u(h^u_{k_{x-1}^{o'}, k_x^{o'}})} < \frac{X + Y - Z}{1 + P_{MD}}, \tag{47}$$

where Δ^u denotes the gap of the uplink transmission time associated with two ordered transferred data in G. Note that $X + Y$ is a function with respect to the local CPU frequencies $f_i^l, i \in \{k_s^o, \ldots, k_n^o\} \bigcap \Psi(o')$ and can achieve minimum when $f_i^l = \min\{\sqrt[3]{\frac{1}{2\kappa}}, f_{peak}\}, \forall i \in \{k_s^o, \ldots, k_n^o\} \bigcap \Psi(o')$. Let $(X+Y)^*$ denote the minimum of $X + Y$. Thus, (47) can be rewritten as

$$\Delta^u < \frac{(X + Y)^* - Z}{1 + P_{MD}}. \tag{48}$$

- As shown in Fig. 14(b), suppose that the tasks in $\{k_s^o, \ldots, k_n^o\}$, which also exist in path o', are the last z tasks offloaded to the edge in path o' under one-climb scheme, i.e., $\{k_s^o, \ldots, k_n^o\} \bigcap \Psi(o') = \{k_{y-z}^{o'}, k_{y-z+1}^{o'}, \ldots, k_y^{o'}\}$. Similarly, if $\hat{T}_{o'} > \tilde{T}_{o'}$ and $\hat{E}_{o'} > \tilde{E}_{o'}$, we have

$$\Delta^d = \frac{O_{k_{y-z-1}^{o'}, k_{y-z}^{o'}}}{R^d(h^d_{k_{y-z-1}^{o'}, k_{y-z}^{o'}})} - \frac{O_{k_y^{o'}, k_{y+1}^{o'}}}{R^d(h^d_{k_y^{o'}, k_{y+1}^{o'}})} < X + Y - Z, \tag{49}$$

where Δ^d denotes the gap of the downlink transmission time associated with two ordered transferred data in G. Then, we have

$$\Delta^d < (X + Y)^* - Z. \tag{50}$$

- As shown in Fig. 14(c), suppose that the tasks in $\{k_s^o, \ldots, k_n^o\}$, which the path o' consists of, are the total tasks offloaded to the edge in path o' under one-climb scheme, i.e., $\{k_s^o, \ldots, k_n^o\} \bigcap \Psi(o') = \{k_x^{o'}, \ldots, k_y^{o'}\}$. If $\hat{T}_{o'} > \tilde{T}_{o'}$ and $\hat{E}_{o'} > \tilde{E}_{o'}$ hold, we have

$$\Delta^{ud} = (1 + P_{MD})\frac{O_{k_{x-1}^{o'}, k_x^{o'}}}{R^u(h^u_{k_{x-1}^{o'}, k_x^{o'}})} + \frac{O_{k_y^{o'}, k_{y+1}^{o'}}}{R^d(h^d_{k_y^{o'}, k_{y+1}^{o'}})} < X + Y - Z. \tag{51}$$

That is,

$$\Delta^{ud} < (X + Y)^* - Z. \tag{52}$$

- Otherwise, as shown in Fig. 14(d), we can find that changing the offloading decisions for the tasks $\{k_s^o, \ldots, k_n^o\} \bigcap \Psi(o')$ from 1 to 0 will lead to multi-time offloading in the path o'. According to the above discussion, we have $\hat{T}_{o'} > \tilde{T}_{o'}$ and $\hat{E}_{o'} > \tilde{E}_{o'}$.

Overall, if $\hat{T}_{o'} > \tilde{T}_{o'}$ and $\hat{E}_{o'} > \tilde{E}_{o'}$, (48), (50) and (52) need to hold. Suppose that we have $\hat{T}_{o'} > \tilde{T}_{o'}$ and $\hat{E}_{o'} > \tilde{E}_{o'}$. Then,

$$\tilde{FT}^l_{M+1}(\tilde{\mathbf{a}}, \tilde{\mathbf{f}}) < \hat{FT}^l_{M+1}(\hat{\mathbf{a}}, \hat{\mathbf{f}}), \tag{53}$$

where $\tilde{FT}^l_{M+1}$ is the total execution time of the task graph G when all the paths follow the one-climb policy, while $\hat{FT}^l_{M+1}$

is the final delay when the tasks in path o prefer two-time offloading scheme. Meanwhile,

$$\tilde{E}(\tilde{\mathbf{a}}, \hat{\mathbf{f}}) < \hat{E}(\hat{\mathbf{a}}, \hat{\mathbf{f}}), \tag{54}$$

where $\tilde{E}$ denotes the total energy consumption of the task graph G when all the paths follow the one-climb policy, while $\hat{E}$ denotes the total energy consumption when the tasks in path o prefer two-time offloading scheme.

Therefore, we have

$$\begin{aligned}\eta(\hat{\mathbf{a}}, \hat{\mathbf{f}}) &= \beta_t \hat{FT}^l_{M+1}(\hat{\mathbf{a}}, \hat{\mathbf{f}}) + \beta_e \hat{E}(\hat{\mathbf{a}}, \hat{\mathbf{f}}) \\ &> \beta_t \tilde{FT}^l_{M+1}(\tilde{\mathbf{a}}, \hat{\mathbf{f}}) + \beta_e \tilde{E}(\tilde{\mathbf{a}}, \hat{\mathbf{f}}) \\ &\geq \beta_t \tilde{FT}^l_{M+1}(\tilde{\mathbf{a}}, \tilde{\mathbf{f}}) + \beta_e \tilde{E}(\tilde{\mathbf{a}}, \tilde{\mathbf{f}}) = \eta(\tilde{\mathbf{a}}, \tilde{\mathbf{f}}),\end{aligned} \tag{55}$$

where the last inequality means that the optimal $\{\hat{\mathbf{f}}\}$ in a two-time offloading scheme is a feasible solution in the one-climb offloading scheme of (P2). Therefore, it contradicts the assumption. To sum up, we have (48), (50) and (52) if the one-climb policy is optimal.

REFERENCES

[1] J. Yan, S. Bi, L. Huang, and Y. J. Zhang, "Deep reinforcement learning based offloading for mobile edge computing with general task graph," in *Proc. IEEE ICC*, Jun. 2020, pp. 1–7.

[2] 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.

[3] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[4] S. Bi and Y. J. Zhang, "Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading," *IEEE Trans. Wireless Commun.*, vol. 17, no. 6, pp. 4177–4190, Jun. 2018.

[5] F. Wang, J. Xu, X. Wang, and S. Cui, "Joint offloading and computing optimization in wireless powered mobile-edge computing systems," *IEEE Trans. Wireless Commun.*, vol. 17, no. 3, pp. 1784–1797, Mar. 2018.

[6] C. You, K. Huang, and H. Chae, "Energy efficient mobile cloud computing powered by wireless energy transfer," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 5, pp. 1757–1771, May 2016.

[7] W. Zhang, Y. Wen, K. Guan, D. Kilper, H. Luo, and D. O. Wu, "Energy-optimal mobile cloud computing under stochastic wireless channel," *IEEE Trans. Wireless Commun.*, vol. 12, no. 9, pp. 4569–4581, Sep. 2013.

[8] M.-H. Chen, B. Liang, and M. Dong, "Joint offloading decision and resource allocation for multi-user multi-task mobile cloud," in *Proc. IEEE ICC*, May 2016, pp. 1–6.

[9] T. Q. Thinh, J. Tang, Q. D. La, and T. Q. S. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Trans. Commun.*, vol. 65, no. 8, pp. 3571–3584, Aug. 2017.

[10] C. You, K. Huang, H. Chae, and B.-H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Trans. Wireless Commun.*, vol. 16, no. 3, pp. 1397–1411, Mar. 2017.

[11] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Trans. Commun.*, vol. 64, no. 10, pp. 4268–4282, Oct. 2016.

[12] Y.-K. Kwok and I. Ahmad, "Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 7, no. 5, pp. 506–521, May 1996.

[13] W. Zhang, Y. Wen, and D. O. Wu, "Collaborative task execution in mobile cloud computing under a stochastic wireless channel," *IEEE Trans. Wireless Commun.*, vol. 14, no. 1, pp. 81–93, Jan. 2015.

[14] 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, Jul. 2018.

[15] S. E. Mahmoodi, R. N. Uma, and K. P. Subbalakshmi, "Optimal joint scheduling and cloud offloading for mobile applications," *IEEE Trans. Cloud Comput.*, vol. 7, no. 2, pp. 301–313, Apr. 2019.

[16] C. Tang *et al.*, "A mobile cloud based scheduling strategy for industrial Internet of Things," *IEEE Access*, vol. 6, pp. 7262–7275, 2018.

[17] S. Guo, B. Xiao, Y. Yang, and Y. Yang, "Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing," in *Proc. IEEE INFOCOM*, Apr. 2016, pp. 1–9.

[18] 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.

[19] M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu, and W. Zhuang, "Learning-based computation offloading for IoT devices with energy harvesting," *IEEE Trans. Veh. Technol.*, vol. 68, no. 2, pp. 1930–1941, Feb. 2019.

[20] 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.

[21] L. Huang, S. Bi, and Y. J. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Trans. Mobile Comput.*, early access, pp. 1–14, Jul. 2019, doi: 10.1109/TMC.2019.2928811.

[22] 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.

[23] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. 33rd Int. Conf. Mach. Learn.*, New York, NY, USA, Jun. 2016, pp. 1928–1937.

[24] K. D. Vogeleer, G. Memmi, P. Jouvelot, and F. Coelho, "The energy/frequency convexity rule: Modeling and experimental validation on mobile devices," in *Proc. Int. Conf. Parallel Process. Appl. Math. (PPAM)*, Warsaw, Poland, Sep. 2013, pp. 793–803.

[25] T. D. Burd and R. W. Broderson, "Processor design for portable systems," *J. VLSI Signal Process. Syst.*, vol. 13, pp. 203–221, Aug./Sep. 1996.

[26] M. Ehrgott, *Multicriteria Optimization*. New York, NY, USA: Springer, 2006.

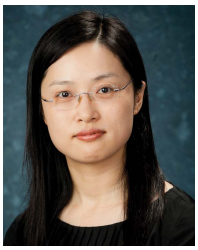
[27] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2004.

[28] E. Perahia and D. C. Cox, "Shadow fading correlation between uplink and downlink," in *Proc. IEEE VTS 53rd Veh. Technol. Conf.*, vol. 1, May 2001, pp. 308–312.

[29] L. P. Qian, Y. J. A. Zhang, and M. Chiang, "Distributed nonconvex power control using Gibbs sampling," *IEEE Trans. Commun.*, vol. 60, no. 12, pp. 3886–3898, Dec. 2012.

Jia Yan (Student Member, IEEE) received the B.Eng. degree from the School of Electronic and Information Engineering, South China University of Technology, Guangzhou, China, in 2017. He is currently pursuing the Ph.D. degree in information engineering with The Chinese University of Hong Kong, Hong Kong. His research interests include optimization and machine learning methods, in particular on the applications of these techniques for mobile edge computing and 5G wireless communications and beyond.

Suzhi Bi (Senior Member, IEEE) received the B.Eng. degree in communications engineering from Zhejiang University, Hangzhou, China, in 2009, and the Ph.D. degree in information engineering from The Chinese University of Hong Kong in 2013. From 2013 to 2015, he was a Post-Doctoral Research Fellow with the Department of Electrical and Computer Engineering, National University of Singapore. Since 2015, he has been with the College of Electronic and Information Engineering, Shenzhen University, Shenzhen, China, where he is currently an Associate Professor. His research interests mainly involve in the optimizations in wireless information and power transfer, mobile computing, and smart power grid communications. He was a co-recipient of the IEEE SmartGridComm 2013 Best Paper Award. He received the IEEE ComSoc Asia–Pacific Outstanding Young Researcher Award in 2019, and the Guangdong Province Pearl River Young Scholar Award in 2018. He is an Associate Editor of the IEEE WIRELESS COMMUNICATIONS LETTERS.



Ying-Jun Angela Zhang (Fellow, IEEE) is currently with the Department of Information Engineering, The Chinese University of Hong Kong. Her research interests include mainly wireless communications systems and smart power systems, in particular optimization techniques for such systems. She is a fellow of IET and a Distinguished Lecturer of the IEEE ComSoc. She was a recipient of the Young Researcher Award from The Chinese University of Hong Kong in 2011. She was also a co-recipient of the 2011 IEEE Marconi Prize Paper Award on Wireless Communications, the 2013 IEEE SmartgridComm Best Paper Award, and the 2014 IEEE ComSoc APB Outstanding Paper Award. As the only winner from engineering science, she has won the Hong Kong Young Scientist Award 2006, conferred by the Hong Kong Institution of Science. She served as the Chair of the Executive Editor Committee of the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS from 2018 to 2019. She has also served for many years on the Editorial Boards for the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, the IEEE TRANSACTIONS ON COMMUNICATIONS, and *Security and Communications Networks* (Wiley). She has been on the organizing committees of major IEEE conferences, including ICC, GLOBECOM, SmartgridComm, VTC, CCNC, ICCC, and MASS. She was the Founding Chair of the IEEE ComSoc Technical Committee on Smart Grid Communications.