Deep Reinforcement Learning for Online Offloading in Wireless Powered Mobile-Edge Computing Networks

Liang Huang, Suzhi Bi, and Ying-Jun Angela Zhang

Abstract

Wireless powered mobile-edge computing (MEC) has recently emerged as a promising paradigm to enhance the data processing capability of low-power networks, such as wireless sensor networks and internet of things (IoT). In this paper, we consider a wireless powered MEC network following the binary offloading policy, such that each computation task of wireless devices (WDs) is either executed locally or fully offloaded to an MEC server. Our goal is to acquire an online algorithm under time-varying wireless channels that jointly optimizes task offloading decisions and wireless resource allocation to maximize the data processing capability of the network. In practice, this requires successively solving hard combinatorial optimization problems to address the multi-user correlation in the offloading decisions. The existing approaches are mainly based on either branch-and-bound algorithm or relaxation heuristics, which are limited by the tradeoff between optimality and efficiency. To tackle this problem, we propose in this paper a Deep Reinforcement learning-based Online Offloading (DROO) framework that implements a deep neural network to generate offloading decisions. In particular, the proposed DROO framework does not require any manually labeled training data as the input, and thus completely removes the need of solving combinatorial optimization problems. Besides, it avoids the curse of dimensionality problem encountered by some existing reinforcement learning approaches and is computationally efficient in large-size networks. To further reduce the computational complexity, we propose an adaptive procedure that automatically adjusts the parameters of the DROO algorithm on the fly. Numerical results show that the proposed algorithm can achieve near-optimal performance while significantly decreasing the computation time by more than an order of magnitude compared with existing methods. For example, the complexity is reduced from several seconds to less than 0.1 second in a 30-user network, making real-time and optimal offloading design truly viable even in a fast fading environment.

Index Terms

Mobile-edge computing, wireless power transfer, reinforcement learning, resource allocation.

L. Huang is with the College of Information Engineering, Zhejiang University of Technology, Hangzhou, China 310058, (e-mail: lianghuang@zjut.edu.cn). S. Bi is with the College of Information Engineering, Shenzhen University, Shenzhen, Guangdong, China 518060 (e-mail: bsz@szu.edu.cn). Y-J. A. Zhang is with the Department of Information Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong. (e-mail: yjzhang@ie.cuhk.edu.hk).

I. Introduction

Modern Internet of Things (IoT) technology is fundamentally constrained by limited battery lifetime and low computing power of size-constrained wireless devices (WDs), especially for computation intensive applications such as augmented reality. Thanks to the recent advance in *wireless power transfer* (WPT) technology, the batteries of WDs can be continuously charged over the air without the need of battery replacement [1]. Meanwhile, the device computing power can be effectively enhanced by the recent development of *mobile-edge computing* (MEC) technology [2]. With MEC, the WDs can offload intensive computations to nearby edge servers for reduced computation latency and energy consumption [3].

The newly emerged *wireless powered MEC* combines the advantages of the two aforementioned technologies, whose deployment in IoT networks is promising to solve the two fundamental performance limitations [4], [5]. In this paper, we consider a wireless powered MEC network as shown in Fig. 1, where the access point (AP) is responsible for both transferring RF (radio frequency) energy to and receiving computation offloading from the WDs. In particular, the WDs follow a *binary task offloading* policy [6], which is a commonly used offloading model for non-partitionable simple sensing tasks in IoT networks. In this case, a task is either computed locally or offloaded to the MEC server for remote computing.

In a multi-user scenario, a major challenge is the joint optimization of individual computing mode selection (i.e., offloading or local computing) and transmission time allocation (on WPT and offloading) to achieve the maximum computation performance. Due to the combinatorial nature of computing mode selection, the problem is generally formulated as mixed integer programming (MIP). To tackle the MIP problem, branch-and-bound algorithms [7] and dynamic programming [8] are used to solve for the globally optimal offloading solution. However, the search spaces of both methods increase exponentially with the network size N and are computationally prohibitive for large-scale MEC networks. To reduce computational complexity, heuristic local searching methods are proposed to reduce the computational complexity. For instance, [5] proposed a coordinate descent (CD) method that searches along one binary variable at a time. A similar heuristic search method for multi-server MEC networks was studied in [9], which iteratively adjusts binary offloading decisions. Another widely adopted heuristic is through convex relaxation, e.g., by relaxing integer variables to be continuous between 0 and 1 [10] or by approximating the binary constraints with quadratic constraints [11]. Nonetheless, on one hand, the solution quality of reduced-complexity heuristics is not guaranteed. On the other hand, both searching-based and convex relaxation methods often require considerable number of iterations for an algorithm to reach a satisfying local optimum. Hence, they are unsuitable for real-time processing in fast fading channels, as the optimization problem needs to be re-solved once the channel fading has varied significantly.

Recently, deep reinforcement learning has emerged as an effective method for handling reinforcement learning problems with large state space [12] and action space [13]. In particular, it relies on deep neural networks (DNNs) [14] to learn from the training data samples, and eventually produces the optimal mapping from the state space to the action space. There exists few recent work on deep reinforcement learning-based offloading for MEC networks [15]–[17]. For an energy-harvesting MEC networks, a deep Q-network (DQN) based offloading policy is proposed in [16] to optimize the computation performance of a single WD served by multiple edge servers. Under the similar network setup, an online computation offloading policy based on DQN is studied in [17] under random task arrivals. However, both works consider discretized channel gains as the input state vector, and thus suffer from the curse of dimensionality and slow convergence when high channel quantization accuracy is required. Besides, because of its exhaustive search nature in selecting the action in each iteration, DQN is not suitable for handling problems with high-dimensional action spaces [18]. In our problem, there are a total of 2^N offloading decisions (actions) to choose from, where DQN is evidently inapplicable even for small N, e.g., $N = 20$.

In this paper, we consider a wireless powered MEC network with one AP and multiple WDs as shown in Fig. 1, where each WD follows a binary offloading policy. In particular, we focus on developing an online offloading algorithm to achieve fast optimization under fast fading channels. Towards this end, we propose a deep reinforcement learning-based online offloading (DROO) framework to maximize the weighted sum computation rate of all the WDs, i.e., the number of processed bits within a unit time. Compared to the existing integer programming and learning-based methods, we have the following novel contributions:

1) The proposed DROO framework takes continuous wireless channel gains as the input to the embedded DNN for offloading action generation. Compared to some existing DQN-based methods that use quantized channel gains, DROO method not only enhances the modeling accuracy but also improves the convergence rate when high channel modeling accuracy is required. Besides, to generate an offloading action, the proposed DROO framework only

needs to select from few candidate actions each time. Compared to some conventional DQN methods that require to search the entire action space, DROO is computationally feasible and efficient in large-size networks with high-dimensional action space.

2) Moreover, the proposed DROO framework does not require any labeled data training sample to update its action generating policy. Instead, through repeated interactions with the wireless fading channel, it can learn from the past offloading experience and automatically improve its action generating policy via reinforcement learning. As such, it completely removes the need of solving complex MIP problems, and thus avoids the optimality-efficiency tradeoff encountered by the existing optimal and heuristic methods.
3) Within the proposed framework, we devise a novel order-preserving quantization method for DROO to generate multiple binary offloading actions from the relaxed action produced by the DNN. The proposed quantization method provides higher diversity in the generated actions and leads to better convergence performance than traditional K-nearest-neighbor (KNN) quantization method.
4) We further develop an adaptive procedure that automatically adjusts the parameters of DROO algorithm on the fly. Specifically, it gradually decreases the number of convex resource allocation sub-problems needed to be solved in a time frame. This effectively reduces the computational complexity without compromising the solution quality.

We evaluate the proposed DROO framework under extensive numerical studies. Our results show that on average the DROO algorithm achieves over 99.5% of the computation rate of the existing near-optimal benchmark method [5]. In the meantime, it significantly reduces the CPU time by more than an order of magnitude, e.g., from several seconds to less than 0.1 second in a 30-user network. This makes real-time and optimal design truly viable in wireless powered MEC networks even in a fast fading environment.

The remainder of this paper is organized as follows. In Section II, we describe the system model and problem formulation. We introduce the detailed designs of the DROO algorithm in Section III. Numerical results are presented in Section IV. Finally, the paper is concluded in Section V.

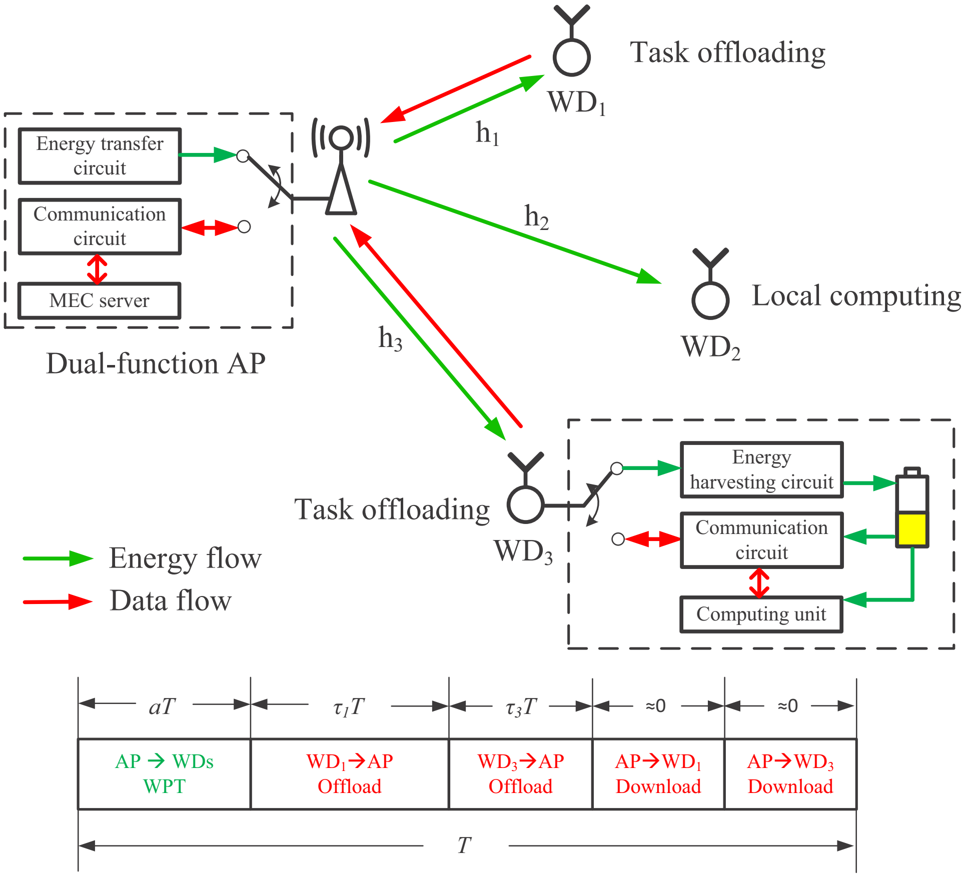

Fig. 1: An example of the considered wireless powered MEC network and system time allocation.

II. Preliminary

A. System Model

As shown in Fig. 1, we consider a wireless powered MEC network consisting of an AP and N fixed WDs, denoted as a set $\mathcal{N} = \{1, 2, \ldots, N\}$, where each device has a single antenna. In practice, this may correspond to a static sensor network. The AP has stable power supply and can broadcast RF energy to the WDs. Each WD has a rechargeable battery that can store the harvested energy to power the operations of the device. The AP is assumed to have higher computation capability than the WDs, so that the WDs may offload their computation tasks to the AP. Specifically, we suppose that WPT and communication (computation offloading) are performed in the same frequency band. Accordingly, a time-division-multiplexing (TDD) circuit is implemented at each device to avoid mutual interference between WPT and communication.

The system time is divided into consecutive time frames of equal length T. Let h_i denote the wireless channel gain between the AP and the i-th WD at a tagged time frame. The channel is assumed to be reciprocal in the downlink and uplink,[1] and remains unchanged within each time frame but may vary across different frames. At the beginning of a time frame, aT amount of time is used for WPT, $a \in [0,1]$, where the AP broadcasts RF energy for the WDs to harvest. Specifically, the i-th WD harvests $E_i = \mu P h_i a T$ amount of energy, where $\mu \in (0,1)$ denotes the energy harvesting efficiency and P denotes the AP transmit power [1]. With the harvested energy, each WD needs to accomplish a computing task before the end of a time frame. In this paper, we consider a binary computation offloading policy, such that the computation is done locally at the WD (such as WD2 in Fig. 1) or offloaded to the AP (such as WD1 and WD3 in Fig. 1). Let $x_i \in \{0,1\}$ be an indicator variable, where $x_i = 1$ denotes that the i-th user's computation task is offloaded to the AP, and $x_i = 0$ denotes that the task is computed locally.

B. *Local Computing Mode*

A WD in the local computing mode can harvest energy and compute its task simultaneously [4]. Let f_i denote the processor's computing speed (cycles per second) and $0 \le t_i \le T$ denote the computation time. Then, the amount of processed bits by the WD is $f_i t_i/\phi$, where $\phi > 0$ denotes the number of cycles needed to process one bit of task data. Meanwhile, the energy consumption of the WD due to the computing is constrained by $k_i f_i^3 t_i \le E_i$, where k_i denotes the computation energy efficiency coefficient [10]. It can be shown that to process the maximum amount of data within T under the energy constraint, a WD should exhaust the harvested energy and compute throughout the time frame, i.e., $t_i^* = T$ and accordingly $f_i^* = \left(\frac{E_i}{k_i T}\right)^{\frac{1}{3}}$. Thus, the local computation rate (in bits per second) is

$$r_{L,i}^*(a) = \frac{f_i^* t_i^*}{\phi T} = \eta_1 \left(\frac{h_i}{k_i}\right)^{\frac{1}{3}} a^{\frac{1}{3}}, \qquad (1)$$

where $\eta_1 \triangleq (\mu P)^{\frac{1}{3}}/\phi$ is a fixed parameter.

C. *Edge Computing Mode*

Due to the TDD structure constraint, a WD in the offloading mode can only offload its task to the AP after harvesting energy. We denote $\tau_i T$ as the offloading time of the i-th WD, $\tau_i \in [0,1]$.

[1] The channel reciprocity assumption is made to simplify the notations of channel state. However, the results of this paper can be easily extended to the case with unequal uplink and downlink channels.

Here, we assume that the computing speed and the transmit power of the AP is much larger than the size- and energy-constrained WDs, e.g., by more than three orders of magnitude. Besides, the computation result to be downloaded to the WD is much shorter than the data offloaded to the edge server. Accordingly, as shown in Fig. 1, we safely neglect the time spent on task computation and download by the AP, such that the WDs only consume energy and time on data offloading (like the same assumptions made in [4], [19]). In this case, the tagged time frame is allocated to WPT and task offloading, i.e.,

$$\sum_{i=1}^{N} \tau_i + a \leq 1. \tag{2}$$

To maximize the computation rate, an offloading WD exhausts its harvested energy on task offloading, i.e., $P_i^* = \frac{E_i}{\tau_i T}$. Accordingly, the computation rate equals to its data offloading capacity, i.e.,

$$r_{O,i}^*(a, \tau_i) = \frac{B\tau_i}{v_u} \log_2 \left(1 + \frac{\mu P a h_i^2}{\tau_i N_0}\right). \tag{3}$$

D. Problem Formulation

Among all the parameters in (1) and (3), we assume that only the wireless channel gains $\mathbf{h} = \{h_i | i \in \mathcal{N}\}$ are time-varying in the considered period, while the others (e.g., w_i's and k_i's) are fixed parameters. Accordingly, the weighted sum computation rate of the wireless powered MEC network in a tagged time frame is denoted as

$$Q\left(\mathbf{h}, \mathbf{x}, \boldsymbol{\tau}, a\right) \triangleq \sum_{i=1}^{N} w_i \left((1 - x_i) r_{L,i}^*(a) + x_i r_{O,i}^*(a, \tau_i)\right), \tag{4}$$

where $\mathbf{x} = \{x_i | i \in \mathcal{N}\}$, $\boldsymbol{\tau} = \{\tau_i | i \in \mathcal{N}\}$, and $w_i > 0$ denotes the weight assigned to the i-th WD.

For each time frame with channel realization $\mathbf{h}$, we are interested in maximizing the weighted sum computation rate:

$$\begin{aligned}
(P1):\ Q^*\left(\mathbf{h}\right) = \underset{\mathbf{x}, \boldsymbol{\tau}, a}{\text{maximize}} \quad & Q\left(\mathbf{h}, \mathbf{x}, \boldsymbol{\tau}, a\right) && \text{(5a)}\\
\text{subject to} \quad & \textstyle\sum_{i=1}^{N} \tau_i + a \leq 1, && \text{(5b)}\\
& a \geq 0,\ \ \tau_i \geq 0,\ \ \forall i \in \mathcal{N}, && \text{(5c)}\\
& x_i \in \{0, 1\}. && \text{(5d)}
\end{aligned}$$

We can easily infer that $\tau_i = 0$ if $x_i = 0$, i.e., when the i-th WD is in the local computing mode.

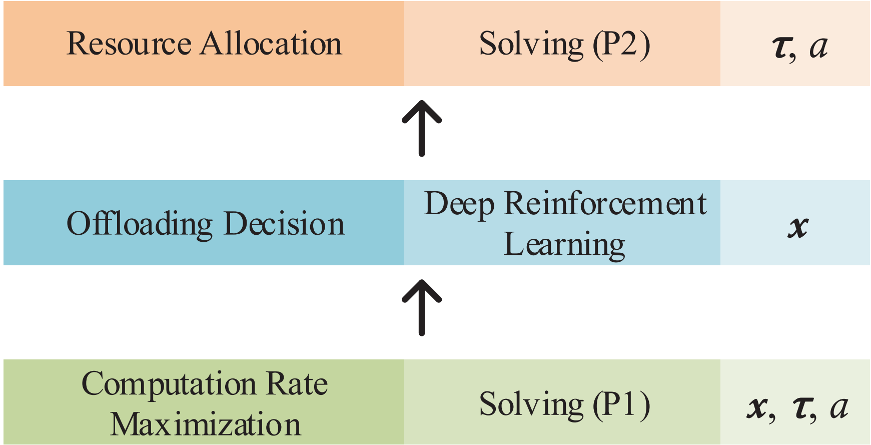

Fig. 2: The two-level optimization structure of solving (P1).

Problem (P1) is a mixed integer programming non-convex problem, which is hard to solve. However, once $\mathbf{x}$ is given, (P1) reduces to a convex problem as follows.

$$\begin{aligned}(P2):\ Q^*(\mathbf{h},\mathbf{x}) = \underset{\boldsymbol{\tau},a}{\text{maximize}} \quad & Q(\mathbf{h},\mathbf{x},\boldsymbol{\tau},a)\\ \text{subject to} \quad & \textstyle\sum_{i=1}^{N}\tau_i + a \leq 1,\\ & a \geq 0,\ \ \tau_i \geq 0,\ \ \forall i \in \mathcal{N}.\end{aligned}$$

Accordingly, problem (P1) can be decomposed into two sub-problems, namely, offloading decision and resource allocation (P2), as shown in Fig. 2:

- *Offloading Decision*: One needs to search among the 2^N possible offloading decisions to find an optimal or a satisfying sub-optimal offloading decision $\mathbf{x}$. For instance, some meta-heuristic searching methods are proposed in [5] and [9] to optimize the offloading decisions. However, due to the extremely large search space, it often takes a large number of searching iterations before convergence.
- *Resource Allocation*: The optimal time allocation $\{a^*, \boldsymbol{\tau}^*\}$ of the convex problem (P2) can be efficiently solved, e.g., using a one-dimensional bi-section search over the dual variable associated with the time allocation constraint in $O(N)$ complexity [5].

The major difficulty of solving (P1) lies in the offloading decision problem. Traditional optimization algorithms require iteratively adjusting offloading decisions towards the optimum, which is fundamentally infeasible for real-time system optimization under fast fading channel. To tackle the complexity issue, we propose in this paper a novel deep reinforcement learning-based online offloading (DROO) algorithm that can achieve millisecond order of computation time in solving the offloading decision problem.

Before leaving this section, it is worth mentioning that the proposed method is different from some existing deep neural network (DNN) approaches for wireless resource allocation optimization or channel estimation, such as in [20] and [21]. Conventional DNN method is based on supervised learning, which requires a large number of manually labeled training samples (e.g., the $(\mathbf{h}, \mathbf{x})$ pairs in this paper) to achieve high solution quality. In contrast, the proposed method does not require any training samples such that it avoids the need to generate samples by computing the hard offloading-decision problem. Meanwhile, supervised learning methods are often not robust to the change of channel distributions. For instance, if the DNN is trained using data samples generated from a fixed device placement, it needs to be re-trained once some WDs change their locations significantly. The proposed DROO algorithm, however, can automatically update its offloading decision policy once the channel distribution changes and thus is more suitable in dynamic wireless applications.

III. The DROO Algorithm

We aim to devise an offloading policy function π that can quickly generate an optimal offloading action $\mathbf{x}^* \in \{0,1\}^N$ of (P1) once the channel realization $\mathbf{h} \in \mathbb{R}^N_{>0}$ is revealed at the beginning of each time frame. The policy is denoted as

$$\pi : \mathbf{h} \mapsto \mathbf{x}^*. \tag{6}$$

The proposed DROO algorithm gradually generates such policy function π after repeated interactions with the wireless powered MEC system.

A. Algorithm Overview

The structure of the algorithm is illustrated in Fig. 3. The DROO algorithm is composed of two alternating stages: offloading action generation and offloading policy update. The generation of offloading action relies on the use of a DNN, which is characterized by its embedded parameters θ, e.g., the weights that connect the hidden neurons. In the t-th time frame, the DNN takes the channel gain $\mathbf{h}_t$ as the input, and outputs a relaxed offloading action $\hat{\mathbf{x}}_t$ (each entry is relaxed to continuous between 0 and 1) based on its current offloading policy π_{θ_t}, parameterized by θ_t. The relaxed action is then quantized into K binary offloading actions, where one best action $\mathbf{x}_t^*$ is

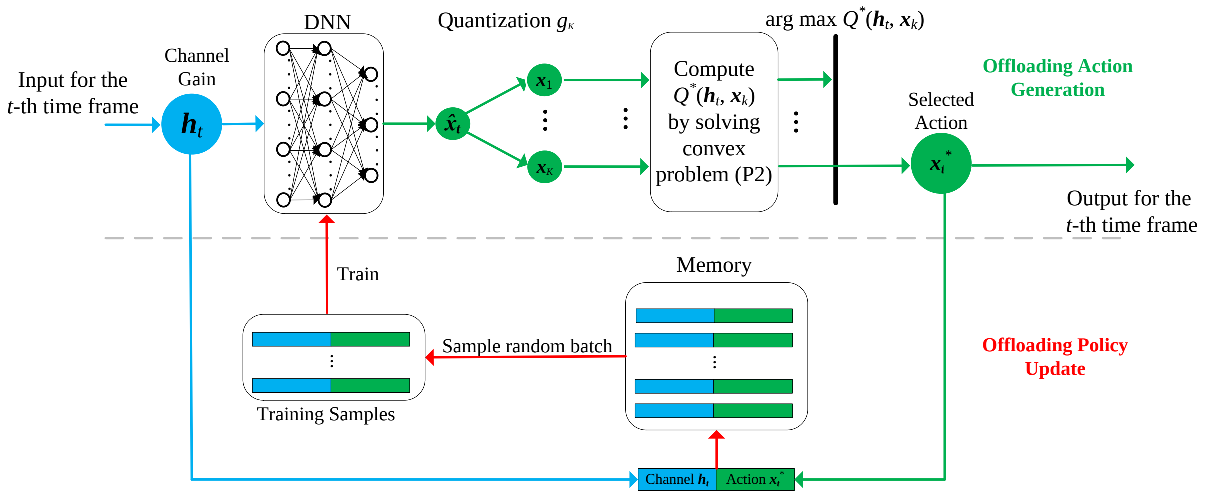

Fig. 3: The schematics of the proposed DROO algorithm.

selected based on the achievable computation rate as in (P2). The network takes the offloading action $\mathbf{x}_t^*$, receives a reward $Q^*(\mathbf{h}_t, \mathbf{x}_t^*)$, and adds the newly obtained state-action pair $(\mathbf{h}_t, \mathbf{x}_t^*)$ to a memory.

Subsequently, in the policy update stage of the t-th time frame, a batch of training samples are drawn from the memory to train the DNN. After training, the DNN will update its parameter from θ_t to θ_{t+1} (and equivalently the offloading policy $\pi_{\theta_{t+1}}$). The new offloading policy $\pi_{\theta_{t+1}}$ will be used in the next time frame to generate offloading decision $\mathbf{x}_{t+1}^*$ according to the new channel $\mathbf{h}_{t+1}$ observed. Such a reinforcement learning iteration will repeat thereafter as new channel realizations are observed, and the DNN will continuously improve its policy π_{θ_t} (and the quality of offloading decisions generated). The descriptions of the two stages are detailed as the following subsections.

B. Offloading Action Generation

Suppose that we observe the channel gain realization $\mathbf{h}_t$ in the t-th time frame, where $t = 1, 2, \cdots$. The parameters of the DNN θ_t are randomly initialized following a zero-mean normal distribution when $t = 1$. The DNN first outputs a relaxed computation offloading action $\hat{\mathbf{x}}_t$, represented by a parameterized function $\hat{\mathbf{x}}_t = f_{\theta_t}(\mathbf{h}_t)$, where

$$\hat{\mathbf{x}}_t = \{\hat{x}_{t,i} | \hat{x}_{t,i} \in [0, 1], i = 1, \cdots, N\} \tag{7}$$

and $\hat{x}_{t,i}$ denotes the i-th entry of $\hat{\mathbf{x}}_t$.

The well-known universal approximation theorem claims that one hidden layer with a large number of hidden neurons suffices to approximate any continuous mapping f if a proper activation function is applied at the neurons, e.g., sigmoid, ReLu, and tanh functions [22]. Here, we use ReLU as the activation function in the hidden layers, where the output y and input v of a neuron are related by $y = \max\{v, 0\}$. In the output layer, we use a sigmoid activation function, i.e., $y = 1/\left(1 + e^{-v}\right)$, such that the relaxed offloading action satisfies $\hat{x}_{t,i} \in (0, 1)$.

Then, we quantize $\hat{\mathbf{x}}_t$ to obtain K binary offloading actions, where K is a design parameter. The quantization function, g_K, is defined as

$$g_K : \hat{\mathbf{x}} \mapsto \{\mathbf{x}_k \mid \mathbf{x}_k \in \{0, 1\}^N, k = 1, \cdots, K\}. \tag{8}$$

K can be any integer within $[1, 2^N]$, where a larger K results in better solution quality and higher computational complexity, and vice versa. To balance the performance and complexity, we propose an *order-preserving quantization* method that produces at most N binary offloading actions, where N is the number of WDs. The basic idea is to preserve the ordering during quantization. That is, for each quantized action $\mathbf{x}_k$, $x_{k,i} \geq x_{k,j}$ should hold if $\hat{x}_{t,i} \geq \hat{x}_{t,j}$ for all $i, j \in \{1, \cdots, N\}$. Specifically, $\{\mathbf{x}_k\}$ is generated as follows:

1) The first binary offloading decision $\mathbf{x}_1$ is obtained as

$$x_{1,i} = \begin{cases} 1 & \hat{x}_{t,i} > 0.5, \\ 0 & \hat{x}_{t,i} \leq 0.5, \end{cases} \tag{9}$$

for $i = 1, \cdots, N$.

2) For the remaining $K - 1$ actions, we first order the entries of $\hat{\mathbf{x}}$ with respective to their distances to 0.5, denoted by $|\hat{x}_{(1)} - 0.5| \leq |\hat{x}_{(2)} - 0.5| \leq \cdots \leq |\hat{x}_{(i)} - 0.5| \cdots \leq |\hat{x}_{(N)} - 0.5|$, where $\hat{x}_{(i)}$ is denoted as the i-th order statistic of $\hat{\mathbf{x}}$. Then, the k-th offloading decision $\mathbf{x}_k$ is obtained as

$$x_{k,i} = \begin{cases} 1 & \hat{x}_{t,i} > \hat{x}_{(k-1)}, \\ 1 & \hat{x}_{t,i} = \hat{x}_{(k-1)} \text{ and } \hat{x}_{(k-1)} < 0.5, \\ 0 & \hat{x}_{t,i} = \hat{x}_{(k-1)} \text{ and } \hat{x}_{(k-1)} > 0.5, \\ 0 & \hat{x}_{t,i} < \hat{x}_{(k-1)}, \end{cases} \tag{10}$$

for $i = 1, \cdots, N$ and $k = 2, \cdots, K - 1$.

For instance, when $\hat{\mathbf{x}}_t$ = [0.2, 0.4, 0.7, 0.9] and $K = 4$, the 4 offloading actions generated from the above quantization method are $\mathbf{x}_1$ = [0, 0, 1, 1], $\mathbf{x}_2$ = [0, 1, 1, 1], $\mathbf{x}_3$ = [0, 0, 0, 1], and $\mathbf{x}_4$

= [1, 1, 1, 1]. In comparison, when conventional KNN method is used, the obtained actions are $\mathbf{x}_1$ = [0, 0, 1, 1], $\mathbf{x}_2$= [0, 1, 1, 1], $\mathbf{x}_3$ = [0, 0, 0, 1], and $\mathbf{x}_4$ = [0, 1, 0, 1].

Evidently, we can obtain at most N quantized offloading decisions with the order-preserving quantization method. Compared to the KNN method where the quantized solutions are closely placed around $\hat{x}$, the offloading actions produced by the order-preserving quantization method are separated by a larger distance. Intuitively, this creates higher diversity in the candidate action set, thus increasing the chance of finding a local maximum around $\hat{\mathbf{x}}_t$. In Section IV-A, we show that the proposed order-preserving quantization method achieves better convergence performance than KNN method.

Recall that each candidate action $\mathbf{x}_k$ can achieve $Q^*(\mathbf{h}_t, \mathbf{x}_k)$ computation rate by solving (P2). Therefore, the best offloading action $\mathbf{x}_t^*$ at the t-th time frame is chosen as

$$\mathbf{x}_t^* = \arg \max_{\mathbf{x}_i \in \{\mathbf{x}_k\}} Q^*(\mathbf{h}_t, \mathbf{x}_i). \tag{11}$$

Note that the K-times evaluation of $Q^*(\mathbf{h}_t, \mathbf{x}_k)$ can be processed in parallel to speed up the computation of (11). Then, the network takes the offloading action $\mathbf{x}_t^*$ and performs the corresponding optimal resource allocation.

C. *Offloading Policy Update*

The obtained best offloading action will be used to update the offloading policy of the DNN. Specifically, we maintain an initially empty memory of limited capacity. At the t-th time frame, a new training data sample $(\mathbf{h}_t, \mathbf{x}_t^*)$ is added to the memory. When the memory is full, the newly generated data sample replaces the oldest one.

We use the experience replay technique [12], [23] to train the DNN using the stored data samples. In the t-th time frame, we randomly select a batch of training data samples $\{(\mathbf{h}_\tau, \mathbf{x}_\tau^*) \mid \tau \in \mathcal{T}_t\}$ from the memory, characterized by a set of time indices $\mathcal{T}_t$. The parameters θ_t of the DNN are updated by applying the Adam algorithm [24] to reduce the averaged cross-entropy loss, as

$$L(\theta_t) = -\frac{1}{|\mathcal{T}_t|}\sum\nolimits_{\tau \in \mathcal{T}_t}\Big((\mathbf{x}_\tau^*)^\intercal \log f_{\theta_t}(\mathbf{h}_\tau) + (1 - \mathbf{x}_\tau^*)^\intercal \log\big(1 - f_{\theta_t}(\mathbf{h}_\tau)\big)\Big),$$

where $|\mathcal{T}_t|$ denotes the size of $\mathcal{T}_t$, the superscript $\intercal$ denotes the transpose operator, and the log function denotes the element-wise logarithm operation of a vector. The detailed update procedure of the Adam algorithm is omitted here for brevity. In practice, we train the DNN in every δ time frames after collecting sufficient number of new data samples. The experience replay technique used in our framework has several advantages. First, the batch update reduces the complexity of using an entire set of data samples. Second, the reuse of historical data reduces the variance of θ_t during the iterative update. Third, the random sampling fastens the convergence by reducing the correlation in the training samples.

Overall, the DNN gradually learns from the best state-action pairs $(\mathbf{h}_t, \mathbf{x}_t^*)$'s in each time frame. As such, it becomes "smarter" over time and continuously improves its produced offloading decisions. Furthermore, with the finite memory space constraint, the DNN only learns from the most recent data samples with high solution quality. This closed-loop reinforcement learning mechanism gradually approaches the optimal policy π after enough number of iterations. We provide the pseudo-code of the DROO algorithm in Algorithm 1.

Algorithm 1: An online DROO algorithm to solve the offloading decision problem.

```
input  : Wireless channel gain $\mathbf{h}_t$ at each time frame $t$
output: Offloading action $\mathbf{x}_t^*$, and the corresponding optimal resource allocation for each
        time frame $t$;
Initialize the DNN with random parameters $\theta_1$ and empty memory $\mathcal{R}$;
Set iteration number $M$ and the training interval $\delta$;
for $t = 1, 2, \ldots, M$ do
    Generate a relaxed offloading action $\hat{\mathbf{x}}_t = f_{\theta_t}(\mathbf{h}_t)$;
    Quantize $\hat{\mathbf{x}}_t$ into $K$ binary actions $\{\mathbf{x}_k\} = g_K(\hat{\mathbf{x}}_t)$;
    Compute $Q^*(\mathbf{h}_t, \mathbf{x}_k)$ for all $\{\mathbf{x}_k\}$ by solving (P2);
    Select the best action $\mathbf{x}_t^* = \arg\max_{\{\mathbf{x}_k\}} Q^*(\mathbf{h}_t, \mathbf{x}_k)$;
    Update the memory by adding $(\mathbf{h}_t, \mathbf{x}_t^*)$ in $\mathcal{R}$;
    if $t \bmod \delta = 0$ then
        Uniformly sample a batch of data set $\{(\mathbf{h}_\tau, \mathbf{x}_\tau^*) \mid \tau \in \mathcal{T}_t\}$ from the memory;
        Train the DNN with $\{(\mathbf{h}_\tau, \mathbf{x}_\tau^*) \mid \tau \in \mathcal{T}_t\}$ and update $\theta_t$ using the Adam algorithm;
    end
end
```

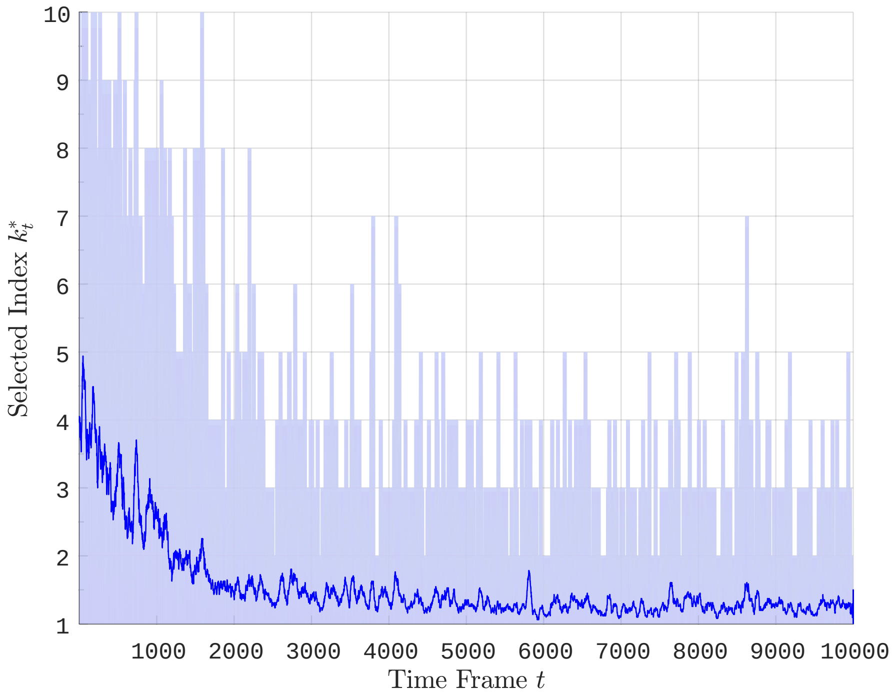

Fig. 4: The index k_t^* of the best offloading actions $\mathbf{x}_t^*$ for DROO algorithm when the number of WDs is $N = 10$ and $K = N$. The detailed simulation setups are presented in Section IV.

D. Adaptive Setting of K

Compared to conventional optimization algorithms, the DROO algorithm has the advantage in removing the need of solving hard MIP problems and thus has the potential to significantly reduce the complexity. The major computational complexity of the DROO algorithm comes from solving (P2) K times in each time frame to select the best offloading action. Evidently, a larger K in general leads to better offloading decision in each time frame and accordingly a better offloading policy in the long term. Therefore, there exists a fundamental performance-complexity tradeoff in setting the value of K.

With the order-preserving quantization method, we generate at most N quantized actions each time, i.e., $K \leq N$. However, using a fixed $K = N$ is not only computationally inefficient but also unnecessary in terms of computation rate performance. To see this, consider a wireless powered MEC network with 10 WDs. We apply the DROO algorithm with $K = 10$ and plot in Fig. 4 the index of the best action $\mathbf{x}_t^*$ calculated from (11) over time, denoted as k_t^*. For instance, $k_t^* = 2$

indicates that the best action in the t-th time frame is ranked the second among the K ordered quantized actions. In the figure, the curve is plotted as the 50-time-frames rolling average of k_t^* and the light shadow region is the upper and lower bounds of k_t^* in the past 50 time frames. Apparently, most of the selected indices k_t^* are no larger than 5 when $t \geq 5000$. This indicates that those generated offloading actions $\mathbf{x}_k$ with $k > 5$ are redundant. In other words, we can gradually reduce K during the learning process to speed up the algorithm without compromising the performance.

In the following, we propose an adaptive method for setting K. We denote K_t as the number of binary offloading actions generated by the quantization function at the t-th time frame. We set $K_1 = N$ initially and update K_t every Δ time frames, where Δ is referred to as the updating interval for K. Upon an update time frame, K_t is set as 1 plus the largest k_t^* observed in the past Δ time frames. The reason for the additional 1 is to allow K_t to increase during the iterations. Mathematically, K_t is calculated as

$$K_t = \begin{cases} N, & t = 1, \\ \min\left(\max\left(k_{t-1}^*, \cdots, k_{t-\Delta}^*\right) + 1, N\right), & t \bmod \Delta = 0, \\ K_{t-1}, & \text{otherwise}, \end{cases}$$

for $t \geq 1$. For an extreme case with $\Delta = 1$, K_t updates in each time frame. Meanwhile, when $\Delta \to \infty$, K_t never updates such that it is equivalent to setting a constant $K = N$. In Section IV-B, we numerically show that setting a proper Δ can effectively speed up the learning process without compromising the computation rate performance.

IV. Numerical Results

In this section, we use simulations to evaluate the performance of the proposed DROO algorithm. In all simulations, we use the parameters of Powercast TX91501-3W with $P = 3$ Watts for the energy transmitter at the AP, and those of P2110 Powerharvester for the energy receiver at each WD.[2] The energy harvesting efficiency $\mu = 0.51$. The distance of the i-th WD to the HAP, denoted by d_i, is uniformly distributed in the range of (2.5,5.2) meters, $i = 1, \cdots, N$. Due to the page limit, the exact values of d_i's are omitted. The average channel gain $\bar{h}_i$ follows the free-space path loss model $\bar{h}_i = A_d \left(\frac{3 \cdot 10^8}{4\pi f_c d_i}\right)^{d_e}$, where $A_d = 4.11$ denotes the antenna gain, $f_c = 915$

[2] See detailed product specifications at http://www.powercastco.com.

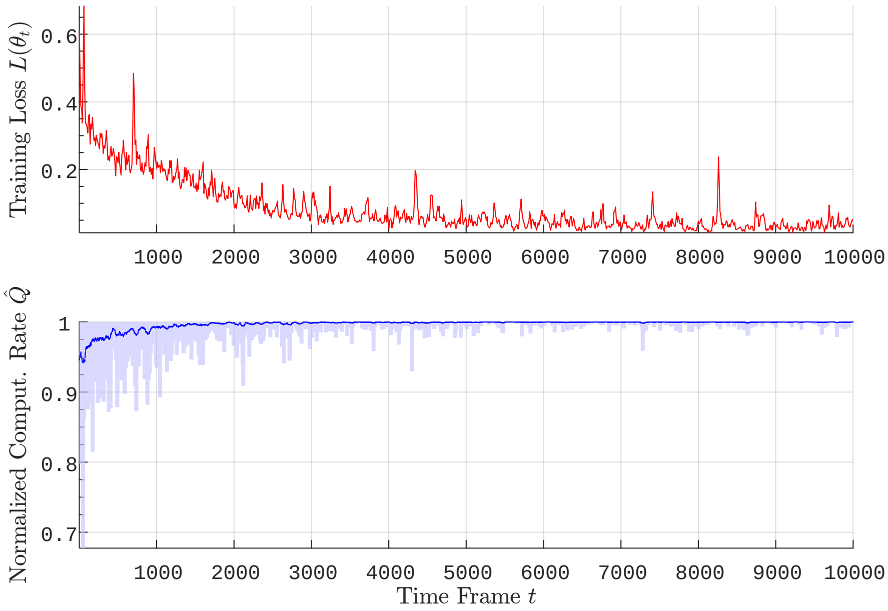

Fig. 5: Normalized computation rates and training losses for DROO algorithm under fading channels when $N = 10$ and $K = 10$.

MHz denotes the carrier frequency, and $d_e = 2.8$ denotes the path loss exponent. The time-varying wireless channel gain of the N WDs at time frame t, denoted by $\mathbf{h}_t = [h_1^t, h_2^t, \cdots, h_N^t]$, is generated according to $h_i^t = \bar{h}_i \alpha_i^t$, where α_i^t is the independent random channel fading factor following an exponential distribution with unit mean. Without loss of generality, the channel gains are assumed to remain the same within one time frame and vary independently from one time frame to another. We assume equal computing efficiency $k_i = 10^{-26}$, $i = 1, \cdots, N$, and $\phi = 100$ for all the WDs [25]. The data offloading bandwidth $B = 2$ MHz, receiver noise power $N_0 = 10^{-10}$, and $v_u = 1.1$. Without loss of generality, we set $T = 1$ and the $w_i = 1$ if i is an odd number and $w_i = 1.5$ otherwise. All the simulations are performed on a desktop with an Intel Core i5-4570 3.2 GHz CPU and 12 GB memory.

We consider a DNN consisting of one input layer, two hidden layers, and one output layer in the proposed DROO algorithm, where the first and second hidden layers have 120 and 80 hidden neurons, respectively. We implement the DROO algorithm in Python with TensorFlow 1.0 and set the training interval $\delta = 10$, training batch size $|\mathcal{T}| = 128$, and memory size as 1024.[3]

[3] The source code is available at https://github.com/revenol/DROO.

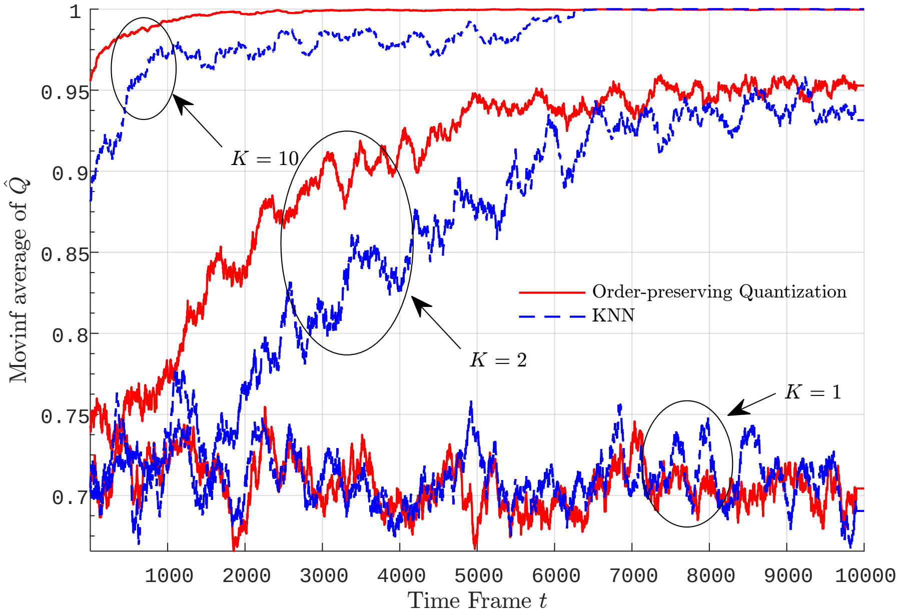

Fig. 6: Moving average of $\hat{Q}$ under different quantization functions and K when $N = 10$.

A. *Convergence Performance*

We first consider a wireless powered MEC network with $N = 10$ WDs. Here, we define the normalized computation rate $\hat{Q}(\mathbf{h}, \mathbf{x}) \in [0, 1]$, as

$$\hat{Q}(\mathbf{h}, \mathbf{x}) = \frac{Q^*(\mathbf{h}, \mathbf{x})}{\max_{\mathbf{x}' \in \{0,1\}^N} Q^*(\mathbf{h}, \mathbf{x}')}, \tag{12}$$

where the optimal solution in the denominator is obtained by enumerating all the 2^N offloading actions.

In Fig. 5, we plot the training loss $L(\theta_t)$ of the DNN and the normalized computation rate $\hat{Q}$. Here, we set a fixed $K = N$. In the figure below, the blue curve denotes the moving average of $\hat{Q}$ over the last 50 time frames, and the light blue shadow denotes the maximum and minimum of $\hat{Q}$ in the last 50 frames. We see that the moving average $\hat{Q}$ of DROO gradually converges to the optimal solution when t is large. Specifically, the achieved average $\hat{Q}$ exceeds 0.98 at an early stage when $t > 400$ and the variance gradually decreases to zero as t becomes larger, e.g., when $t > 3000$. Meanwhile, in the figure above, the training loss $L(\theta_t)$ gradually decreases and stabilizes at around 0.04, whose fluctuation is mainly due to the random sampling of training data.

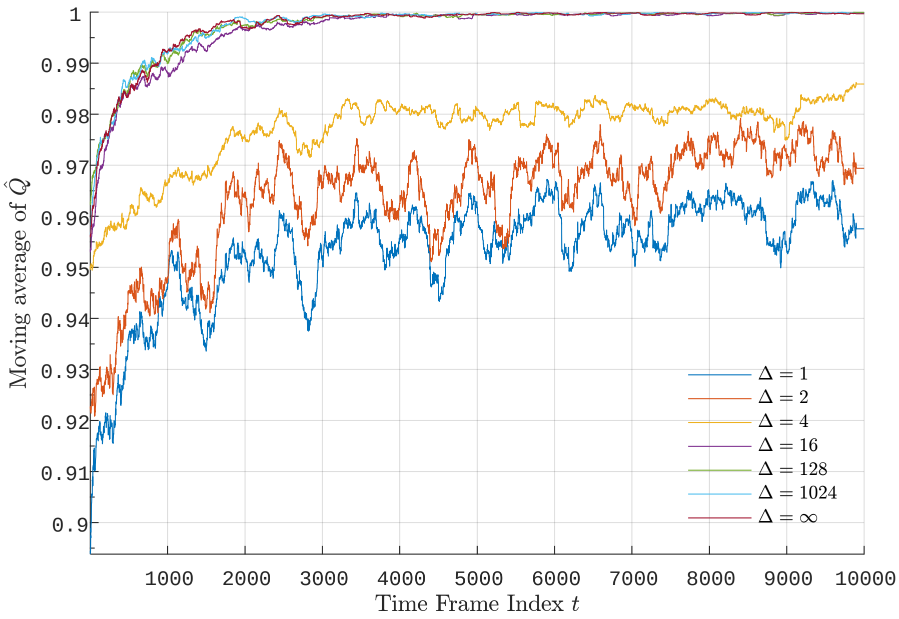

Fig. 7: Moving average of $\hat{Q}$ for DROO algorithm with different updating interval Δ for setting an adaptive K. Here, we set $N = 10$.

In Fig. 6, we compare the performance of two quantization methods: the proposed order-preserving quantization and the conventional KNN quantization method under different K. In particular, we plot the the moving average of $\hat{Q}$ over a window of 200 time frames. When $K = N$, both methods converge to the optimal offloading actions, i.e., the moving average of $\hat{Q}$ approaches 1. However, they both achieve suboptimal offloading actions when K is small. For instance, when $K = 2$, the order-preserving quantization method and KNN both only converge to around 0.95. Nonetheless, we can observe that when $K \geq 2$, the order-preserving quantization method converges faster than the KNN method. Intuitively, this is because the order-preserving quantization method offers a larger diversity in the candidate actions than the KNN method. Therefore, the training of DNN requires exploring less offloading actions before convergence. Notice that the DROO algorithm does not converge for both quantization methods when $K = 1$. This is because the DNN cannot improve its offloading policy when action selection is absent.

The simulation results in this subsection show the effectiveness of the proposed DROO framework can quickly converge to the optimal offloading policy, especially when the proposed order-preserving action quantization method is used.

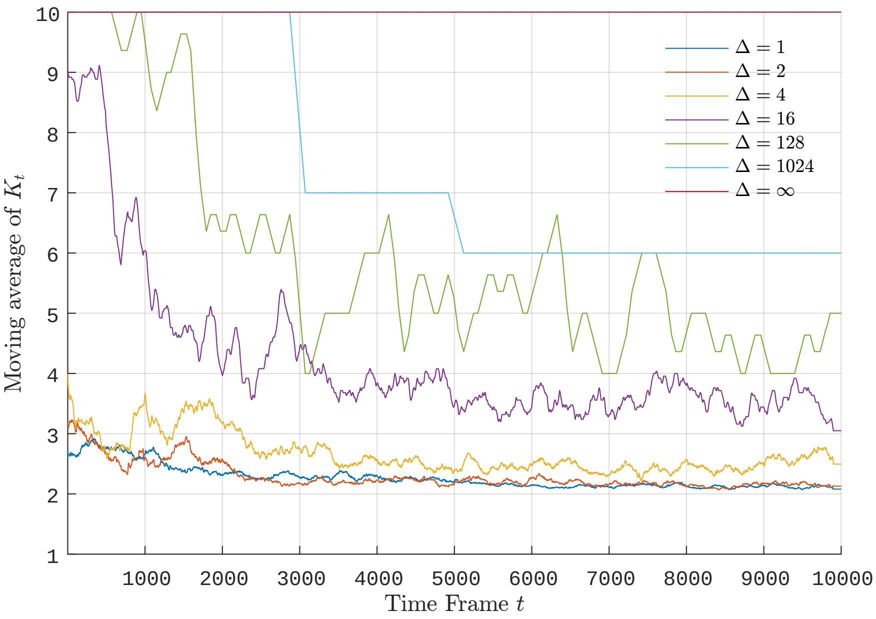

Fig. 8: Dynamics of K_t under different updating interval Δ when $N = 10$.

B. *Impact of Updating Intervals* Δ

In Fig. 7, we further study the impact of the updating interval of K (i.e., Δ) on the convergence property. Here, we use the adaptive setting method of K in Section III.D and plot the moving average of $\hat{Q}$ over a window of 200 time frames. We see that the DROO algorithm converges to the optimal solution only when setting a sufficiently large Δ, e.g., $\Delta \geq 16$. Meanwhile, we also plot in Fig. 8 the moving average of K_t under different Δ. We see that K_t increases with Δ when t is large. This indicates that setting a larger Δ will lead to higher computational complexity, i.e., requires computing (P2) more times in a time frame. Therefore, a performance-complexity tradeoff exists in setting Δ.

To properly choose an updating interval Δ, we plot in Fig. 9 the tradeoff between the total CPU time of 10000 channel realizations and the moving average of $\hat{Q}$ in the last time frame. On one hand, we see that the average of $\hat{Q}$ quickly increases from 0.96 to close to 1 when $\Delta \leq 16$, while the improvement becomes marginal afterwards when we further increase Δ. On the other hand, the CPU time increases monotonically with Δ. To balance between performance and complexity, we set $\Delta = 32$ for DROO algorithm in the following simulations.

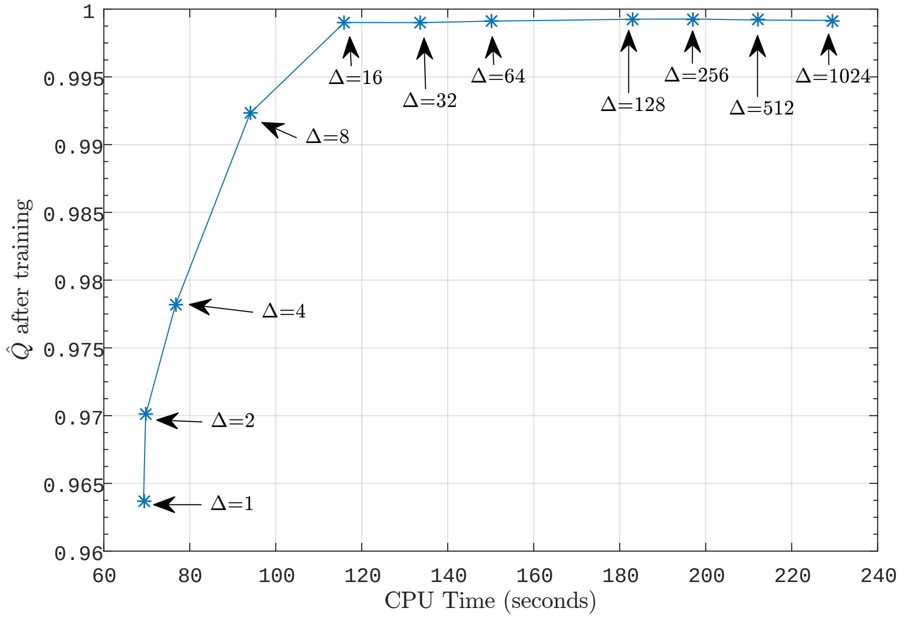

Fig. 9: Tradeoff between $\hat{Q}$ and CPU time after training DROO for 10,000 channel realizations under different updating intervals Δ when $N = 10$.

C. *Computation Rate Performance*

Regarding to the weighted sum computation rate performance, we compare our DROO algorithm with three representative benchmarks:

- *Coordinate Descent (CD) algorithm* [5]. The CD algorithm iteratively swaps in each round the computing mode of the WD that leads to the largest computation rate improvement. That is, from $x_i = 0$ to $x_i = 1$, or vice versa. The iteration stops when the computation performance cannot be further improved by the computing mode swapping. The CD method is shown to achieve near-optimal performance under different N.
- *Local Computing*. All N WDs only perform local computation, i.e., setting $x_i = 0,\ i = 1, \cdots, N$ in (P2).
- *Edge Computing*. All N WDs offload their tasks to the AP, i.e., setting $x_i = 1,\ i = 1, \cdots, N$ in (P2).

In Fig. 10, we first compare the computation rate performance achieved by different offloading algorithms under varying number of WDs, N. Before the evaluation, DROO has been trained with $24,000$ independent wireless channel realizations, and its offloading policy has converged.

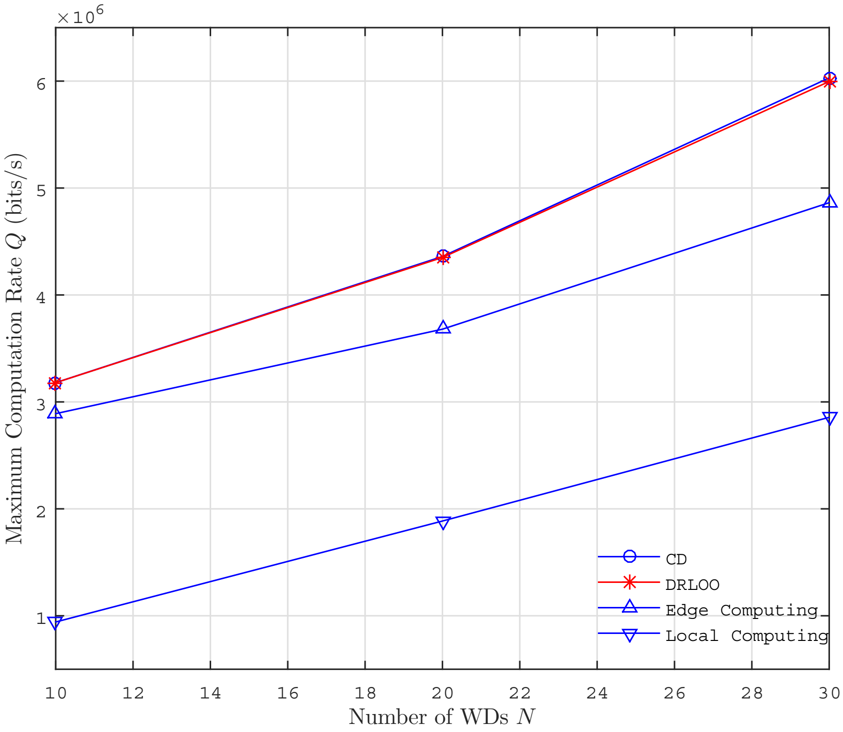

Fig. 10: Comparisons of computation rate performance for different offloading algorithms.

Each point in the figure is the average performance of $6,000$ independent wireless channel realizations. We see that DROO achieves similar near-optimal performance with the CD method, and significantly outperforms the Edge Computing and Local Computing algorithms. In Fig. 11, we further evaluate the performance of the DROO algorithm. For better exposition, we plot the normalized computation rate $\hat{Q}$ achievable by DROO. Specifically, we enumerate all 2^N possible offloading actions as in (12) when $N = 10$. For $N = 20$ and 30, it is computationally prohibitive to enumerate all the possible actions. In this case, $\hat{Q}$ is obtained by normalizing the computation rate achievable by DROO against that of CD method. We then plot both the median and the confidence intervals of $\hat{Q}$ over 6000 independent channel realizations. We see that the median of DROO is always close-to-1 for different number of users, and the confidence intervals are mostly above 0.99. The results in Fig. 10 and 11 show that the proposed DROO method can achieve near-optimal computation rate performance under different network placements.

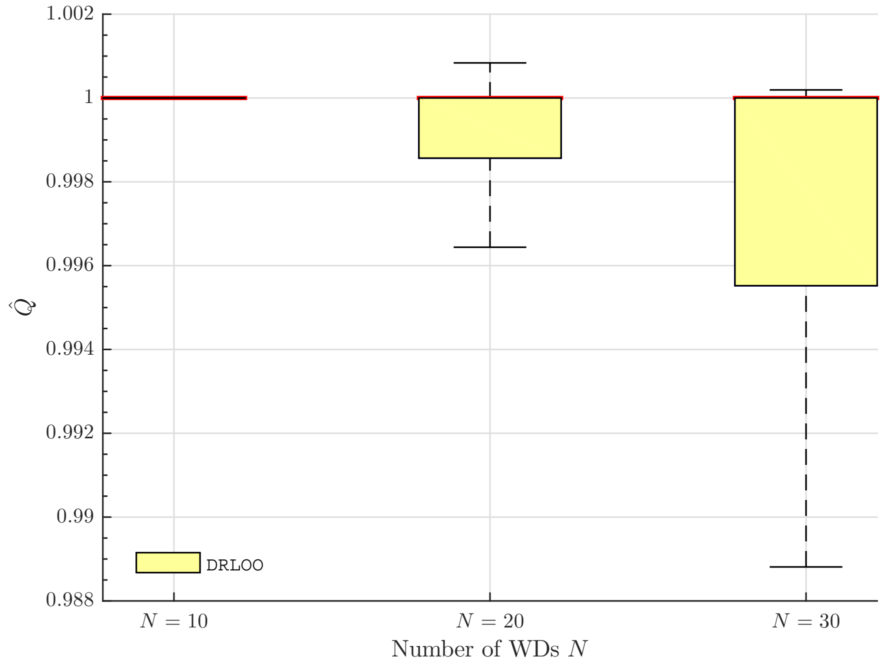

Fig. 11: Boxplot of the normalized computation rate $\hat{Q}$ for DROO algorithm under different number of WDs. The central mark (in red) indicates the median, and the bottom and top edges of the box indicate the 25th and 75th percentiles, respectively.

D. Algorithm Complexity

At last, we evaluate the computation complexity of the DROO algorithm. The computational complexity of DROO algorithm greatly depends on the complexity in solving the resource allocation sub-problem (P2). For fair comparison, we use the same bi-section search method as the CD algorithm in [5]. The CD method is reported to achieve an $O(N^3)$ complexity. For the DROO algorithm, we consider both using a fixed K and an adaptive K as in Section III.D. We see from Table I that the use of an adaptive K can effectively reduce the CPU time than a fixed $K = N$. Besides, DROO with an adaptive K requires much shorter CPU time than the CD method. In particular, it generates an offloading action in less than 0.1 second when $N = 30$, while CD takes 65 times longer CPU time. Overall, DROO achieves similar rate performance as CD algorithm but requires substantially less CPU time. This makes real-time offloading and resource allocation truly viable for wireless powered MEC networks in fading environment.

TABLE I: Comparisons of CPU time

# of WDs	DROO (Fixed $K = N$)	DROO (Adaptive K with Δ = 32)	CD
10	3.6e-2s	1.2e-2s	2.0e-1s
20	1.3e-1s	3.0e-2s	1.3s
30	3.1e-1s	5.9e-2s	3.8s

V. Conclusion

In this paper, we have proposed a deep reinforcement learning-based online offloading algorithm, DROO, to maximize the weighted sum computation rate in wireless powered MEC networks with binary computation offloading. The algorithm does not necessitate any manually labeled training data and learns from the past offloading experience to improve its offloading action produced by a DNN via reinforcement learning. An order-preserving quantization and an adaptive parameter setting method are devised to achieve fast algorithm convergence. Compared to conventional optimization methods, the proposed DROO algorithm completely removes the need of solving hard mixed integer programming problems. Simulation results show that DROO achieves similar near-optimal performance as existing benchmark methods, but reduces the CPU time by more than an order of magnitude, making real-time system optimization truly viable for wireless powered MEC networks in fading environment.

References

[1] S. Bi, C. K. Ho, and R. Zhang, "Wireless powered communication: Opportunities and challenges," *IEEE Commun. Mag.*, vol. 53, no. 4, pp. 117–125, Apr. 2015.

[2] M. Chiang and T. Zhang, "Fog and IoT: An overview of research opportunities," *IEEE Internet Things J.*, vol. 3, no. 6, pp. 854–864, Dec. 2016.

[3] 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.

[4] 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.

[5] S. Bi and Y. J. A. 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.

[6] 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, Aug. 2017.

[7] P. M. Narendra and K. Fukunaga, “A branch and bound algorithm for feature subset selection,” *IEEE Trans. Comput.*, vol. C-26, no. 9, pp. 917–922, Sep. 1977.

[8] D. P. Bertsekas, *Dynamic programming and optimal control*. Athena Scientific Belmont, MA, 1995, vol. 1, no. 2.

[9] T. X. Tran and D. Pompili, “Joint task offloading and resource allocation for multi-server mobile-edge computing networks,” *arXiv preprint arXiv:1705.00704*, 2017.

[10] 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.

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

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, p. 529, Feb. 2015.

[13] G. Dulac-Arnold, R. Evans, H. van Hasselt, P. Sunehag, T. Lillicrap, J. Hunt, T. Mann, T. Weber, T. Degris, and B. Coppin, “Deep reinforcement learning in large discrete action spaces,” *arXiv preprint arXiv:1512.07679*, 2015.

[14] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, p. 436, May 2015.

[15] Y. He, F. R. Yu, N. Zhao, V. C. Leung, and H. Yin, “Software-defined networks with mobile edge computing and caching for smart cities: A big data deep reinforcement learning approach,” *IEEE Commun. Mag.*, vol. 55, no. 12, pp. 31–37, Dec. 2017.

[16] M. Min, D. Xu, L. Xiao, Y. Tang, and D. Wu, “Learning-based computation offloading for IoT devices with energy harvesting,” *arXiv preprint arXiv:1712.08768*, 2017.

[17] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, “Performance optimization in mobile-edge computing via deep reinforcement learning,” *arXiv preprint arXiv:1804.00514*, 2018.

[18] 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. ICLR*, 2016.

[19] 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.

[20] H. Sun, X. Chen, Q. Shi, M. Hong, X. Fu, and N. D. Sidiropoulos, “Learning to optimize: Training deep neural networks for wireless resource management,” in *Proc. IEEE SPAWC*, Jul. 2017, pp. 1–6.

[21] H. Ye, G. Y. Li, and B. H. Juang, “Power of deep learning for channel estimation and signal detection in OFDM systems,” *IEEE Wireless Commun. Lett.*, vol. 7, no. 1, pp. 114–117, Feb 2018.

[22] S. Marsland, *Machine learning: an algorithmic perspective*. CRC press, 2015.

[23] L.-J. Lin, “Reinforcement learning for robots using neural networks,” Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, Tech. Rep., 1993.

[24] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *Proc. ICLR*, 2015.

[25] 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.