Performance Optimization in Mobile-Edge Computing via Deep Reinforcement Learning

Xianfu Chen, Honggang Zhang, Celimuge Wu, Shiwen Mao, Yusheng Ji, and Mehdi Bennis

Abstract—To improve the quality of computation experience for mobile devices, mobile-edge computing (MEC) is emerging as a promising paradigm by providing computing capabilities within radio access networks in close proximity. Nevertheless, the design of computation offloading policies for a MEC system remains challenging. Specifically, whether to execute an arriving computation task at local mobile device or to offload a task for cloud execution should adapt to the environmental dynamics in a smarter manner. In this paper, we consider MEC for a representative mobile user in an ultra dense network, where one of multiple base stations (BSs) can be selected for computation offloading. The problem of solving an optimal computation offloading policy is modelled as a Markov decision process, where our objective is to minimize the long-term cost and an offloading decision is made based on the channel qualities between the mobile user and the BSs, the energy queue state as well as the task queue state. To break the curse of high dimensionality in state space, we propose a deep Q-network-based strategic computation offloading algorithm to learn the optimal policy without having a priori knowledge of the dynamic statistics. Numerical experiments provided in this paper show that our proposed algorithm achieves a significant improvement in average cost compared with baseline policies.

I. Introduction

With the proliferation of smart devices, more and more mobile applications, such as location-based virtual/augmented reality and online gaming, are emerging and gaining popularity [1]. However, the mobile devices are in general resource-constrained, for example, the battery capacity and the local CPU computation power are limited. The tension between computation-intensive applications and resource-constrained mobile devices creates a hurdle of having satisfactory Quality-of-Service (QoS) and Quality-of-Experience (QoE), and is hence driving a revolution in terms of computing infrastructure [2].

Mobile-edge computing (MEC) is envisioned as a promising paradigm to address the hurdle by providing computing capabilities within radio access networks (RANs) in close proximity to mobile users (MUs) [3], [4]. By offloading computation tasks to the resource-rich MEC servers, not only the

X. Chen is with the VTT Technical Research Centre of Finland, Finland (e-mail: xianfu.chen@vtt.fi). H. Zhang is with the College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou, China (e-mail: honggangzhang@zju.edu.cn). C. Wu is with the Graduate School of Informatics and Engineering, University of Electro-Communications, Tokyo, Japan (email: clmg@is.uec.ac.jp). S. Mao is with the Department of Electrical and Computer Engineering, Auburn University, Auburn, AL, USA (email: smao@ieee.org). Y. Ji is with the Information Systems Architecture Research Division, National Institute of Informatics, Tokyo, Japan (e-mail: kei@nii.ac.jp). M. Bennis is with the Centre for Wireless Communications, University of Oulu, Finland (email: bennis@ee.oulu.fi).

computation QoS and QoE can be greatly improved, but the capabilities of mobile devices can be augmented for running resource-demanding applications. Recently, lots of efforts have been centered on the design of computation offloading policy. In [5], Wang et al. developed an alternating direction method of multipliers-based algorithm to resolve the issue of revenue maximization by optimizing computation offloading decisions, resource allocation and content caching strategies. In [6], Hu et al. proposed a two-phase based method for joint power and time allocation while considering cooperative computation offloading in a wireless power transfer-assisted MEC system. The policies in these works are primarily based on one-shot optimization and fail to characterize long-term computation offloading performance.

For a MEC system, the computation offloading process requires wireless data transmission, for which the design of computation offloading policies should take into account the existing environmental dynamics, such as the time-varying channel quality and the task arrival and energy status at a mobile device. In [7], Liu et al. formulated the problem of delay-optimal computation task offloading under a Markov decision process (MDP) framework and developed an efficient one-dimensional search algorithm to find the optimal solution. The challenge for the work in [7] lies in the dependence on statistical information of channel quality variations and task arrivals. In [8], Mao et al. investigated a dynamic computation offloading policy for a MEC system with wireless energy harvesting-enabled mobile devices using Lyapunov optimization techniques. However, the Lyapunov optimization can only construct an approximately optimal solution. Xu et al. developed in [9] a reinforcement learning based algorithm to learn the optimal computation offloading policy, which at the same time does not need a priori knowledge of environmental statistics.

When the MEC meets an ultra dense RAN, a number of base stations (BSs) are available with different data transmission qualities. In this context, the explosion in state space makes the conventional reinforcement learning algorithms [9]–[11] infeasible. The focus of this paper is to consider the MEC in an ultra dense system, where the mobile devices are wireless charging enabled. The problem of designing an optimal computation offloading policy is formulated as a MDP. We resort to a deep neural network based function approximator to deal with the curse of state space explosion [12]. As a major contribution, we propose an online strategic computation offloading policy based on a deep Q-network (DQN), with which a typical MU
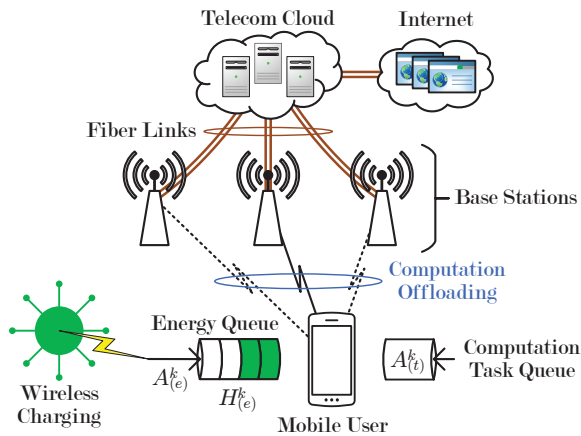
Fig. 1. Illustration of a mobile-edge computing system with wireless charging enabled mobile devices.

in the ultra dense MEC system is able to realize a significant performance improvement.

The rest of the paper is organized as follows. In the next section, we describe the system model and the basic assumptions considered in this paper. In Section III, we formulate the problem of designing an optimal computation offloading policy as a MDP. We detail the proposed algorithm in Section IV. To validate the proposed study, we provide numerical experiments under various settings in Section V. Finally, we draw the conclusions in Section VI.

II. SYSTEM MODEL AND ASSUMPTIONS

As depicted in Fig. 1, we shall consider in this paper an ultra dense networking environment covered by a set $\mathcal{N} = \{1, \cdots, N\}$ of BSs. The BSs are connected via the fiber cables to a resource-rich computing infrastructure, namely, the telecom cloud, which is deployed by the network operator. By strategically offloading the computation tasks for cloud execution, the wireless charging enabled MUs can expect a significantly improved computation experience. In the analysis that follows, we focus on a representative MU in the dense RAN. The time horizon is discretized into epochs, each of which is of equal duration δ (in seconds) and is indexed by an integer $k \in \mathbb{N}_+$. The whole system operates over a common spectrum, and we denote the frequency bandwidth by W (in Hz).

We denote the channel gain state between the MU and a BS $n \in \mathcal{N}$ during each epoch k as g_n^k, which independently picks a value from a finite state space $\mathcal{G}_n$. The channel state transitions across the epochs are modelled as a finite-state discrete-time Markov chain. We let μ (in bits) represent the input data size of a computation task. The computation task arrivals at the MU are assumed to be an independent and identically distributed sequence of Bernoulli random variables with a common parameter $\lambda_{(t)} \in [0, 1]$. More specifically, we choose $A_{(t)}^k \in \{0, 1\}$ as the task arrival indicator, that is, $A_{(t)}^k = 1$ if a task is generated at the beginning of epoch k and otherwise, $A_{(t)}^k = 0$. Then, $\mathsf{Pr}\{A_{(t)}^k = 1\} = 1 - \mathsf{Pr}\{A_{(t)}^k = 0\}$

$0\} = \lambda_{(t)}$, where $\mathsf{Pr}\{\cdot\}$ denotes the probability of an event. In our considered MEC system, the computation task can be either executed locally at the mobile device of the MU or offloaded to and executed at the telecom cloud. At the beginning of each epoch k, the MU makes a joint decision regarding computation offloading $T^k \in \{-1\} \cup \{0\} \cup \mathcal{N}$ and energy allocation E^k (in energy units). Note that $T^k = -1$ is the case the MU decides not to execute the computation task, and there will be no computation task execution delay and $E^k = 0$, hence leading to the dropping of an arrived task.

We have $T^k = 0$ if the computation task is executed locally at the mobile device of the MU during an epoch k. Let ν be the number of CPU cycles required to process one input bit of the computation task. Then the allocated CPU-cycle frequency at the MU can be calculated as

$$f_{(l)}^k = \sqrt{\frac{E^k}{\tau \mu \nu}}, \quad (1)$$

with the given energy E^k, where τ is the effective switched capacitance that depends on chip architecture of the mobile device [13]. Moreover, the CPU-cycle frequency is constrained by $f_{(l)}^k \leq \overline{f}$. The incurred delay for local computation execution at epoch k can be hence expressed by

$$b_{(l)}^k = \frac{\mu \nu}{f_{(l)}^k}. \quad (2)$$

At the beginning of epoch k, if the MU decides to offload the computation task to the telecom cloud for execution via a BS $n \in \mathcal{N}$, namely, $T^k = n$, the input data should be first transmitted to the cloud. In a dense networking scenario, the achievable data rate can be written as

$$R^k = W \log_2 \left(1 + I^{-1} g_n^k \frac{E^k}{b_{(c),(tr)}^k} \right), \quad (3)$$

where I is the received average power of interference and additive background noise, while $b_{(c),(tr)}^k$ is the transmission time and is the solution of

$$R^k b_{(c),(tr)}^k = \mu. \quad (4)$$

In (4) above, we suppose that the energy E^k is evenly assigned to the input bits of the computation task [14]. After receiving the input bits of the offloaded computation task, the telecom cloud proceeds to execute it. We let $f_{(c)}$ be the constant CPU-cycle frequency assigned to the MU, which is based on the subscribed cloud-computing contract between the MU and the network operator. The execution time of the computation task at the cloud takes up to

$$b_{(c),(ex)} = \frac{\mu \nu}{f_{(c)}}. \quad (5)$$

Therefore, the overall delay resulted from offloading computation task for cloud execution is

$$b_{(c)}^k = b_{(c),(tr)}^k + b_{(c),(ex)}, \quad (6)$$

where as in the existing works [8], [15], we neglect the time consumed for sending the computation outcomes from the telecom cloud back to the MU.

Let H^k be the energy queue length of the MU at the beginning of epoch k, which evolves according to

$$H^{k+1} = \min\left\{H^k - E^k + A_{(e)}^k, \overline{H}\right\}, \qquad (7)$$

where $\overline{H}$ limits the maximum number of energy units that can be stored and $A_{(e)}^k$ is the number of energy units acquired from the wireless environment by the end of epoch k.

III. PROBLEM FORMULATION

The computation task arrivals from the MU can be offloaded to the telecom cloud depending on the channel qualities, the energy queue state and the computation task queue state. We denote $\mathbf{x}^k = (A_{(t)}^k, H^k, \mathbf{g}^k) \in \mathcal{X} = \{0, 1\} \times \{0, 1, \cdots, \overline{H}\} \times \{\times_{n \in \mathcal{N}} \mathcal{G}_n\}$ as the network state of the MU at each epoch k, where $\mathbf{g}^k = (g_n^k, n \in \mathcal{N})$. With observation $\mathbf{x}^k$ at the beginning of epoch k, the MU strategically decides an action $\mathbf{y}^k = (T^k, E^k) \in \mathcal{Y} = \{\{-1\} \cup \{0\} \cup \mathcal{N}\} \times \{0, 1, \cdots, \overline{H}\}$ following a stationary control policy $\mathbf{\Phi} = (\Phi_{(t)}, \Phi_{(e)})$, where $\Phi_{(t)}$ and $\Phi_{(e)}$ are, respectively, the computation offloading and the energy allocation policies. That is, $\mathbf{\Phi}(\mathbf{x}^k) = (\Phi_{(t)}(\mathbf{x}^k), \Phi_{(e)}(\mathbf{x}^k)) = (T^k, E^k)$. Given $\mathbf{\Phi}$, the $\{\mathbf{x}^k : k \in \mathbb{N}_+\}$ is a controlled Markov chain with the state transition probability as below,

$$\Pr\{\mathbf{x}^{k+1}|\mathbf{x}^k, \mathbf{\Phi}(\mathbf{x}^k)\} = \left(\prod_{n \in \mathcal{N}} \Pr\{g_n^{k+1}|g_n^k\}\right) \Pr\{A_{(t)}^{k+1}\}$$
$$\times \Pr\{H^{k+1}|H^k, \mathbf{\Phi}(\mathbf{x}^k)\}. \qquad (8)$$

When the MU is associated with a BS at epoch k, which is different from the previous one, additional handover delay is incurred. We assume that the energy consumption during the handover procedure is negligible for the MU and the delay during the occurrence of one handover is ζ (in seconds). Then the handover delay $b_{(h)}^k = b_{(h)}(\mathbf{x}^k, \mathbf{y}^k)$ at epoch k is

$$b_{(h)}(\mathbf{x}^k, \mathbf{y}^k) = \zeta \mathbb{1}_{\{\langle T^k \in \mathcal{N}\rangle \wedge \langle T^j \in \mathcal{N}\rangle \wedge \langle T^k \neq T^j\rangle\}}, \qquad (9)$$

where $\mathbb{1}_{\{\Omega\}}$ is an indicator function that equals 1 if condition Ω is met and otherwise, 0, and $j = \max\{\ell : T^\ell \in \mathcal{N}, \ell \in \mathbb{N}_+, \ell < k\}$. The experienced delay is the key performance indicator for evaluating the quality of a task computing experience. In addition, due to the sporadic nature of energy units that can be received across the epochs, the newly arriving computation tasks at an epoch may have to be dropped, the cost $b_{(d)}^k = b_{(d)}(\mathbf{x}^k, \mathbf{y}^k)$ of which is defined to be

$$b_{(d)}(\mathbf{x}^k, \mathbf{y}^k) = \Pr\{A_{(t)}^k = 1\} \mathbb{1}_{\{T^k = -1\}}. \qquad (10)$$

In line with the discussions in previous sections, we define the task execution cost $p^k = p(\mathbf{x}^k, \mathbf{y}^k)$ at each epoch k as the weighted sum of the execution delay, the handover delay and the computation task dropping cost, namely,

$$p(\mathbf{x}^k, \mathbf{y}^k) = b(\mathbf{x}^k, \mathbf{y}^k) + \rho b_{(h)}^k + \varphi b_{(d)}^k, \qquad (11)$$

where $\rho, \varphi \in \mathbb{R}_+$ are the weights of the handover delay and the computation task dropping cost, respectively, and

$$b(\mathbf{x}^k, \mathbf{y}^k) = \begin{cases} 0, & \text{if } T^k = -1; \\ b_{(l)}^k, & \text{if } T^k = 0; \\ b_{(c)}^k, & \text{otherwise.} \end{cases} \qquad (12)$$

Taking expectation with respect to the per-epoch task execution costs over the randomized network states $\mathbf{x}^k$ and the actions $\mathbf{y}^k$ induced by a given control policy $\mathbf{\Phi}$, the expected long-term cost of the MU conditioned on an initial network state $\mathbf{x}^1$ can be expressed as

$$V(\mathbf{x}, \mathbf{\Phi}) = \mathsf{E}_{\mathbf{\Phi}}\left[(1 - \gamma)\sum_{k=1}^{\infty}(\gamma)^{k-1}p^k|\mathbf{x}^1 = \mathbf{x}\right], \qquad (13)$$

where $\mathbf{x} = (A_{(t)}, H, \mathbf{g})$ with $\mathbf{g} = (g_n : n \in \mathcal{N})$, $\gamma \in [0, 1)$ is the discount factor, and $(\gamma)^{k-1}$ denotes the discount factor to the $(k-1)$-th power. The objective of the MU is to design an optimal control policy $\mathbf{\Phi}^* = (\Phi_{(t)}^*, \Phi_{(e)}^*)$ that minimizes $V(\mathbf{x}, \mathbf{\Phi})$, for any given initial network state $\mathbf{x}$, which can be formally formulated as

$$\mathbf{\Phi}^* = \arg\min_{\mathbf{\Phi}} V(\mathbf{x}, \mathbf{\Phi}), \forall \mathbf{x} \in \mathcal{X}. \qquad (14)$$

We denote $V(\mathbf{x}) = V(\mathbf{x}, \mathbf{\Phi}^*)$ as the optimal state-value function, $\forall \mathbf{x} \in \mathcal{X}$.

IV. SOLVING THE OPTIMAL CONTROL POLICY

The formulated computation offloading optimization in (14) is in essential a single-agent infinite-horizon MDP with the discounted cost criterion. In this section, we shall first investigate the optimal solution within the conventional MDP framework and then proceed to propose a deep reinforcement learning based scheme with limited network statistics information.

A. Optimal MDP Solution

The optimal state-value function, namely, $V(\mathbf{x}), \forall \mathbf{x} \in \mathcal{X}$, can be achieved by solving the Bellman's optimality equation as in the following lemma [11].

Lemma 1. The optimal state-value function $\{V(\mathbf{x}), \forall \mathbf{x} \in \mathcal{X}\}$ satisfies the Bellman's optimality equation, that is, $\forall \mathbf{x}$,

$$V(\mathbf{x}) =$$
$$\min_{\mathbf{y} \in \mathcal{Y}}\left\{(1 - \gamma)p(\mathbf{x}, \mathbf{y}) + \gamma \sum_{\mathbf{x}' \in \mathcal{X}} \Pr\{\mathbf{x}'|\mathbf{x}, \mathbf{y}\}V(\mathbf{x}')\right\}, \qquad (15)$$

where $p(\mathbf{x}, \mathbf{y})$ is the task execution cost when action $\mathbf{y}$ is performed under network state $\mathbf{x}$ and $\mathbf{x}' = (A_{(t)}', H', \mathbf{g}')$ is the subsequent network state with $\mathbf{g}' = (g_n' : n \in \mathcal{N})$.

Remark 1: The size X of the network state space $\mathcal{X}$ can be calculated as $X = 2 \times (1 + \overline{H}) \times \prod_{n \in \mathcal{N}} |\mathcal{G}_n|$, where $|\mathcal{G}|$ means the cardinality of the set $\mathcal{G}$. It can be observed that X grows exponentially as the number N of BSs increases.

Remark 2: The traditional solutions to (15) are based on the value or the policy iteration [11], which not only need *complete knowledge* of the channel state transition probabilities, the computation task arrival and the received energy unit
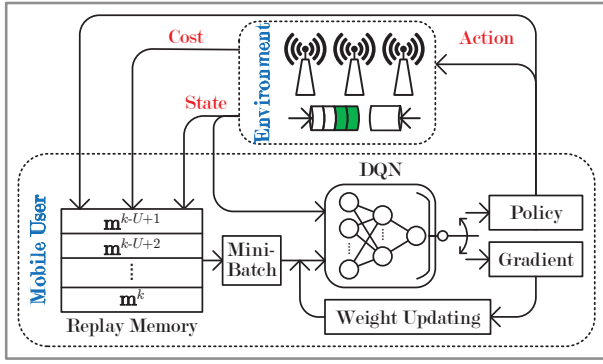
Fig. 2. Deep Q-network (DQN) based mobile-edge computing system.

Algorithm 1 DQN-based Online Strategic Computation Task Offloading

1: **initialize** the replay memory $\mathcal{O}^k$ with a size of U, the mini-batch $\tilde{\mathcal{O}}^k$ with a size of S, and the Q-function with two sets $\boldsymbol{\theta}^k$ and $\hat{\boldsymbol{\theta}}^k$ of random weights, for $k = 1$.
2: **repeat**
3:     At the beginning of epoch k, observe the network state $\mathbf{x}^k \in \mathcal{X}$ and select an action $\mathbf{y}^k \in \mathcal{Y}$ randomly with probability ϵ or $\mathbf{y}^k = \arg\min_{\mathbf{y} \in \mathcal{Y}} Q(\mathbf{x}^k, \mathbf{y}; \boldsymbol{\theta}^k)$ with probability $1 - \epsilon$.
4:     After deploying $\mathbf{y}^k$, observe the cost $p(\mathbf{x}^k, \mathbf{y}^k)$ and the new network state $\mathbf{x}^{k+1} \in \mathcal{X}$.
5:     Store $\mathbf{m}^k = (\mathbf{x}^k, \mathbf{y}^k, p(\mathbf{x}^k, \mathbf{y}^k), \mathbf{x}^{k+1})$ in $\mathcal{O}^k$.
6:     Sample a random mini-batch of transitions $\tilde{\mathcal{O}}^k \subseteq \mathcal{O}^k$.
7:     Update $\boldsymbol{\theta}^{k+1}$ with the gradient given by (19).
8:     Regularly perform $\hat{\boldsymbol{\theta}}^{k+1} = \boldsymbol{\theta}^k$.
9:     Update epoch index by $k \leftarrow k + 1$.
10: **until** A predefined stopping condition is satisfied.

statistics but suffer from *exponential computation complexity* due to the extremely huge network state space even with a reasonable number of BSs. Suppose there is a MEC system with 6 BSs and for each BS, the channel gain is quantized into 8 states. If we set $\overline{H} = 4$ as in the numerical experiments, there are astonishing 2621440 local network states in total for the MU.

The next subsection thereby focuses on developing a practically efficient scheme to approach the optimal policy.

B. Deep Reinforcement Learning

Define the right-hand side of (15) as the optimal action-value function $Q : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$, which is

$$Q(\mathbf{x}, \mathbf{y}) = (1 - \gamma)p(\mathbf{x}, \mathbf{y}) + \gamma \sum_{\mathbf{x}' \in \mathcal{X}} \Pr\{\mathbf{x}'|\mathbf{x}, \mathbf{y}\}V(\mathbf{x}'), \quad (16)$$

$\forall(\mathbf{x}, \mathbf{y}) \in \mathcal{X} \times \mathcal{Y}$, we have $V(\mathbf{x}) = \min_{\mathbf{y} \in \mathcal{Y}} Q(\mathbf{x}, \mathbf{y})$, $\forall \mathbf{x} \in \mathcal{X}$. To address the first technical challenge in Remark 2, we adopt a model-free reinforcement learning scheme called Q-learning [10], which allows us to learn the optimal control policy without any information of dynamic network statistics.

The Q-learning scheme is a simple Q-function update step, which is performed at the beginning of an epoch. Based on the observations of the network state $\mathbf{x}^k$, the action $\mathbf{y}^k$, the received task execution cost $p(\mathbf{x}^k, \mathbf{y}^k)$, the computation task arrival $A_{(t)}^{k+1}$, the number of received energy units $A_{(e)}^k$ at each epoch k, and the resulting network state $\mathbf{x}^{k+1}$ at the next epoch $k + 1$, the MU updates Q-function on-the-fly,

$$Q\big(\mathbf{x}^k, \mathbf{y}^k\big) \leftarrow Q\big(\mathbf{x}^k, \mathbf{y}^k\big) + \quad (17)$$
$$\alpha^k \left((1 - \gamma)p\big(\mathbf{x}^k, \mathbf{y}^k\big) + \gamma \min_{\mathbf{y} \in \mathcal{Y}} Q\big(\mathbf{x}^{k+1}, \mathbf{y}\big) - Q\big(\mathbf{x}^k, \mathbf{y}^k\big) \right),$$

where $\alpha^k \in [0, 1)$ is a time-varying learning rate. It has been proven that if 1) the network state transition probability under the optimal stationary control policy is stationary, 2) $\sum_{k=1}^{\infty} \alpha^k$ is infinite and $\sum_{k=1}^{\infty} (\alpha^k)^2$ is finite, and 3) all state-action pairs are visited infinitely often, the convergence of the Q-learning process is ensured [10]. The last condition can be satisfied if the probability of choosing any action in any network state is non-zero (i.e., *exploration*). Meanwhile, the MU has to exploit the current knowledge in order to perform well (i.e.,

exploitation). A classical way to balance the trade-off between *exploration* and *exploitation* is the ϵ-greedy strategy [11].

Remark 3: The Q-learning rule, which is formulated in (17), relieves the dependence on full network statistics information, but still has to face the curse of a huge network state space, as pointed out in Remark 2.

Hereinafter, we adopt a DQN to online estimate the Q-function [12]. That is, $Q(\mathbf{x}, \mathbf{y}) \approx Q(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta})$, where $(\mathbf{x}, \mathbf{y}) \in \mathcal{X} \times \mathcal{Y}$ and the set of weights is denoted by $\boldsymbol{\theta}$. The proposed DQN-based strategic computation offloading for our considered MEC system is illustrated in Fig. 2. The MU utilizes a replay memory of a finite size U to store the transition $\mathbf{m}^k = (\mathbf{x}^k, \mathbf{y}^k, p(\mathbf{x}^k, \mathbf{y}^k), \mathbf{x}^{k+1})$ that is happened at the end of each epoch k. The memory pool is characterized by $\mathcal{O}^k = \{\mathbf{m}^{k-U+1}, \cdots, \mathbf{m}^k\}$. According to the experience replay technique, the MU randomly samples an experience at each epoch k, i.e., a mini-batch $\tilde{\mathcal{O}}^k \subseteq \mathcal{O}^k$ of S transitions, from $\mathcal{O}^k$ to train the DQN in the direction of minimizing the loss function in (18), where the set of weights of the DQN at an epoch k is denoted as $\boldsymbol{\theta}^k$, and $\hat{\boldsymbol{\theta}}^k$ is a second set of weights for evaluating the action values and is updated to be $\boldsymbol{\theta}^k$ in the next epoch. The gradient of (18) is given by (19). Algorithm 1 summarizes the DQN-based online strategic computation offloading for the MU in a MEC system.

V. NUMERICAL EXPERIMENTS

In this section, we proceed to quantify the performance from our proposed DQN-based online strategic computation offloading.

A. General Setup

For the DQN, the replay memory is assumed to have a capacity of $U = 5000$ and we select the size of the mini-batch as $S = 100$. Throughout the numerical experiments, we suppose there are $N = 6$ BSs in the MEC system connecting the MU with the telecom cloud. The channel gain states

$$L\left(\boldsymbol{\theta}^{k+1}\right) = \mathsf{E}_{(\mathbf{x},\mathbf{y},p(\mathbf{x},\mathbf{y}),\mathbf{x}')\in\tilde{\mathcal{O}}^k}\left[\left((1-\gamma)p(\mathbf{x},\mathbf{y}) + \gamma Q\left(\mathbf{x}', \underset{\mathbf{y}'\in\mathcal{Y}}{\arg\min}Q\left(\mathbf{x}',\mathbf{y}';\hat{\boldsymbol{\theta}}^k\right);\boldsymbol{\theta}^k\right) - Q\left(\mathbf{x},\mathbf{y};\boldsymbol{\theta}^{k+1}\right)\right)^2\right] \quad (18)$$

$$\nabla_{\boldsymbol{\theta}^{k+1}}L\left(\boldsymbol{\theta}^{k+1}\right) =$$
$$\mathsf{E}_{(\mathbf{x},\mathbf{y},p(\mathbf{x},\mathbf{y}),\mathbf{x}')\in\tilde{\mathcal{O}}^k}\left[\left((1-\gamma)p(\mathbf{x},\mathbf{y}) + \gamma Q\left(\mathbf{x}', \underset{\mathbf{y}'\in\mathcal{Y}}{\arg\min}Q\left(\mathbf{x}',\mathbf{y}';\hat{\boldsymbol{\theta}}^k\right);\boldsymbol{\theta}^k\right) - Q\left(\mathbf{x},\mathbf{y};\boldsymbol{\theta}^{k+1}\right)\right)\nabla_{\boldsymbol{\theta}^{k+1}}Q\left(\mathbf{x},\mathbf{y};\boldsymbol{\theta}^{k+1}\right)\right] \quad (19)$$

between the MU and the BSs are from a common finite set $\{-18, -16, -14, -12, -10, -8, -6, -4\}$ (dB), the transitions of which happen across the epochs following respective randomly generated matrices. Each energy unit corresponds to 5×10^{-5} J, and the energy units harvested from the wireless environment follow a Poisson arrival process with average arrival rate $\lambda_{(e)}$. We set $\rho\zeta = 0.9\delta$ and $\beta = 9\delta$. In addition, $\gamma = 0.9$, $W = 10^6$ Hz, $I = 10^{-3}$ W, $\delta = 10^{-3}$ second, $\overline{H} = 4$, $\mu = 10^3$ bits, $\tau = 10^{-28}$, $\nu = 600$ cycles per bit, $\overline{f} = 1.9$ GHz, and $f_{(c)} = 3.9$ GHz. For comparisons, we simulate three baselines as well, namely,

1) *Local* – Whenever a computation task arrives, the MU executes it at the local mobile device using the queued energy units.
2) *Cloud* – All arriving computation tasks are offloaded to the telecom cloud for computing via the BSs with the best channel qualities.
3) *Greedy* – When the computation task queue as well as the energy queue are not empty at an epoch k, the MU decides to execute the task locally or at the cloud to achieve the minimum current delay, i.e., $\min\{b_{(l)}^k, b_{(c)}^k\}$.

B. Experimental Results

We carry out numerical experiments under various settings to validate the proposed work.

1) Experiment 1 – Convergence performance: In this experiment, the goal is to validate the convergence property of our proposed DQN-based online computation task offloading algorithm. We set $\lambda_{(t)} = 0.6$ and $\lambda_{(e)} = 0.5$. The DQN consists of one hidden layer of 128 neurons. In Fig. 3, we plot the simulated variations in the loss function defined as in (18), which reveals that the convergence of our proposed algorithm can be ensured. Based on the convergence of loss function, each result in the following experiments is obtained from one system configuration running for 9×10^5 epochs.

2) Experiment 2 – Performance under different DQN structures: This experiment tries to demonstrate the MEC performance for the MU in terms of the average cost per epoch using a DQN with different numbers of layers and neurons structures. We choose $\lambda_{(t)} = 0.4$ and $\lambda_{(e)} = 0.8$ in simulations for the MU. The results are exhibited in Fig. 4. In the upper plot, the number of neurons per hidden layer is fixed to be 64. It can be observed that a deeper DQN leads to worse average cost performance. The reason is that over a limited
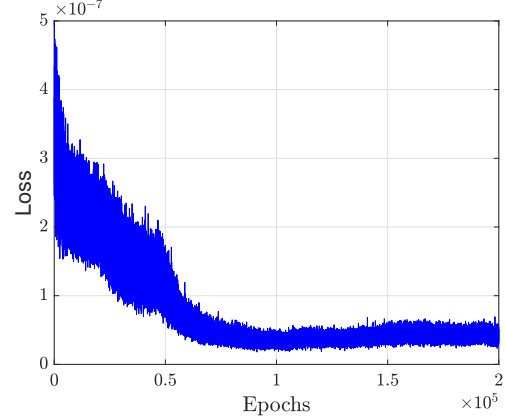

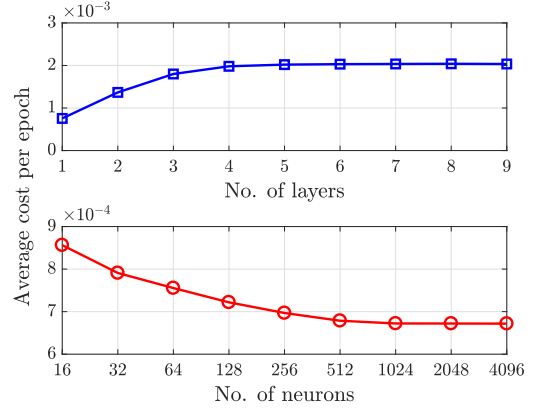Fig. 3. Illustration of convergence property of our proposed algorithm.

Fig. 4. Average cost per epoch versus numbers of layers and neurons.

time horizon, adding more hidden layers to the DQN leads to higher training errors [16]. In the lower plot, only one hidden layer is implemented in the DQN. From the curve, better performance is achieved with a bigger number of neurons. In our considered MEC scenario, a wider (NOT deeper) DQN can better approximate the Q-function.

3) Experiment 3 – Performance with changing $\lambda_{(t)}$ and $\lambda_{(e)}$: We do this experiment to simulate the average performance achieved from the proposed DQN-based algorithm and other three baselines versus the average energy arrival rates. With the findings from Experiment 2, we configure a DQN of
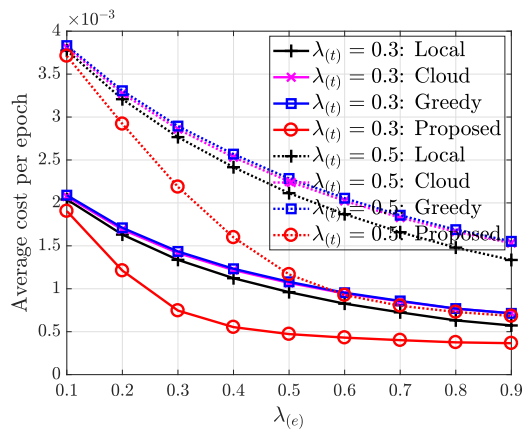
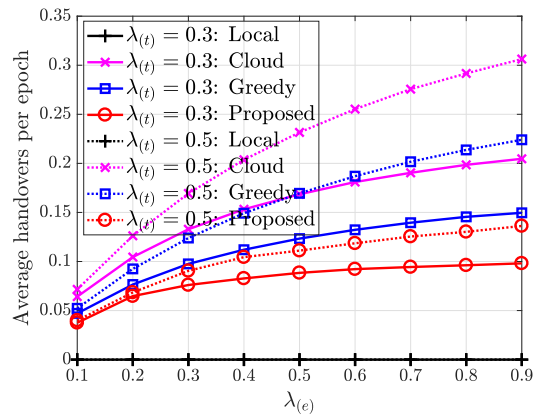Fig. 5. Average cost per epoch versus average energy unit arrival rate.

Fig. 7. Average handovers per epoch versus average energy unit arrival rate.
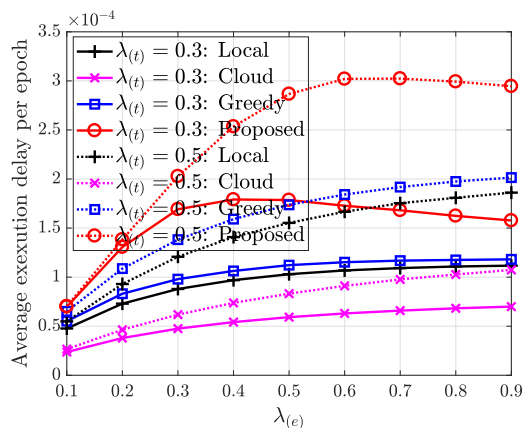

Fig. 6. Average execution delay per epoch versus average energy unit arrival rate.
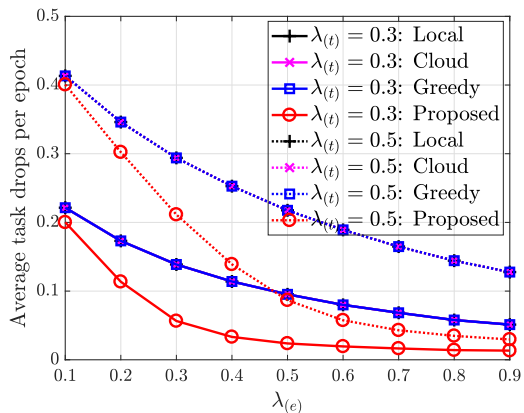

Fig. 8. Average task drops per epoch versus average energy unit arrival rate.

one hidden layer with 512 neurons. The per epoch averages of cost, execution delay, handovers and task drops under $\lambda_{(t)} = 0.3$ and $\lambda_{(t)} = 0.5$ across the entire learning period are depicted in Figs. 5, 6, 7 and 8. From Fig. 5, we can clearly see that compared to the baselines, our proposed algorithm achieves a significant performance improvement in average cost, up to 56%. A higher task arriving probability indicates a longer average delay for executing more computation tasks, more handovers between BSs and more task drops, hence a higher average cost. As the number of energy arrivals increases, the average cost decreases. This is a result of less computation task drops, which dominate the cost function for the weight choices. Interestingly, the increase in energy arrivals does not necessarily reduce the task execution delay and the handovers. This can be explained by the fact that more energy arrivals provide more opportunities for the MU to select a BS with better channel gain to execute a computation task, rather than simply drop it.

VI. CONCLUSIONS

In this paper, we put our emphasis on investigating the design of a smart computation offloading policy for a MU in an ultra dense network by taking into account the dynamics generated from time-varying channel qualities between the MU and the BSs, harvested energy units and task arrivals. To solve the formulated MDP, we propose a DQN-based online strategic computation offloading algorithm that survives the curse of high dimensionality in state space and needs no a priori information of dynamics statistics. We find from numerical experiments that compared to three baselines, our proposed algorithm can achieve minimum long-term cost, up to 56% in performance improvement, which indicates an optimal tradeoff among the computation task execution delay, the handover delay and the task dropping cost.

REFERENCES

[1] "Cisco visual networking index: Global mobile data traffic forecast update, 2016–2021," White Paper, Cisco, Feb. 2017.

[2] M. Satyanarayanan, "The emergence of edge computing," *IEEE Comput.*, vol. 50, no. 1, pp. 30–39, Jan. 2017.

[3] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and Ccomputation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, Q3 2017.

[4] C.-F. Liu, M. Bennis, and H. V. Poor, "Latency and reliability-aware task offloading and resource allocation for mobile edge computing," in *Proc. IEEE GLOBECOM WKSHP*, Singapore, Dec. 2017.

[5] C. Wang, C. Liang, F. R. Yu, Q. Chen, and L. Tang, "Computation offloading and resource allocation in wireless cellular networks with mobile edge computing," *IEEE Trans. Wireless Commun.*, vol. 16, no. 8, pp. 4924–4938, Aug. 2017.

[6] X. Hu, K.-K. Wong, and K. Yang, "Wireless powered cooperation-assisted mobile edge computing," *IEEE Trans. Wireless Commun.*, Early Access Article, 2018.

[7] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, 'Delay-optimal computation task scheduling for mobile-edge computing systems," in *Proc. IEEE ISIT*, Barcelona, Spain, Jul. 2016.

[8] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3590–3605, Dec. 2016.

[9] J. Xu, L. Chen, and S. Ren, "Online learning for offloading and autoscaling in energy harvesting mobile edge computing," *IEEE Trans. Cogn. Commun. Netw.*, vol. 3, no. 3, pp. 361–373, Jul. 2017.

[10] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, no. 3–4, pp. 279–292, May 1992.

[11] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[13] T. D. Burd and R. W. Brodersen, "Processor design for portable systems," *J. VLSI Signal Process. Syst.*, vol. 13, no. 2–3, pp. 203–221, Aug. 1996.

[14] J. Yang and S. Ulukus, "Optimal packet scheduling in an energy harvesting communication system," *IEEE Trans. Commun.*, vol. 60, no. 1, pp. 220–230, Jan. 2012.

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

[16] K. He, X. Zhang, S. Ren, J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE CVPR*, Las Vegas, NV, Jun. 2016.