2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)

When Learning Joins Edge: Real-time Proportional Computation Offloading via Deep Reinforcement Learning

Ning Chen[1], Sheng Zhang[1], Zhuzhong Qian[1], Jie Wu[2], and Sanglu Lu[1]
[1]State Key Lab. for Novel Software Technology, Nanjing University, CN
[2]Department of Computer and Information Sciences, Temple University, Philadelphia, USA
Email: ningc@smail.nju.edu.cn, {sheng, qzz, sanglu}@nju.edu.cn, jiewu@temple.edu

***Abstract*—Computation offloading makes sense to the interaction between users and compute-intensive applications. Current researches focused on deciding locally or remotely executing an application, but ignored the specific offloading proportion of application. A full offloading cannot make the best use of client and server resources. In this paper, we propose an innovative reinforcement learning (RL) method to solve the proportional computation problem. We consider a common offloading scenario with time-variant bandwidth and heterogeneous devices, and the device generates applications constantly. For each application, the client has to choose locally or remotely executing this application, and determines the proportion to be offloaded. We formalize the problem as a long-term optimization problem, and then propose a RL-based algorithm to solve it. The basic idea is to estimate the benefit of posible decisions, of wihch the decision with the maximum benefit is selected. Instead of adopting the original Deep Q Network (DQN), we propose Advanced DQN (ADQN) by adding Priority Buffer Mechanism and Expert Buffer Mechanism, which improves the utilization of samples and overcomes the cold start problem, respectively. The experimental results show ADQN's high feasibility and efficiency compared with several traditional policies, such as None Offloading Policy, Random Offloading Policy, Link Capacity Optimal Policy, and Computing Capability Optimal Policy. At last, we analyse the effect of expert buffer size and learning rate on ADQN's performance.**

***Index Terms*—Computation offloading, Advanced Deep Q Network, Expert Buffer Mechanism**

I. INTRODUCTION

The rise of 5G has greatly strengthened the connection between humans and machines. Meanwhile, compute-intensive and delay-sensitive applications, such as interactive gaming, image/video processing, augmented/virtual reality and face recognition, are becoming popular on mobile devices. Thanks to the emergence of Mobile Edge Computing (MEC) [1]–[3], the data and computation are pushed away from centralized cloud computing infrastructures to the logical edge of a network, thereby enabling analytics and knowledge generation to occur closer to the mobile users. MEC enhances the computation capability at the edge of mobile networks by deploying high-performance edge servers.

Although computation offloading has been extensively studied, there is still no unified method to solve this problem, which is largely due to the heterogeneity of edge devices and the time-variant nature of the network. Existing researches on computation offloading only consider the tradeoff between cost

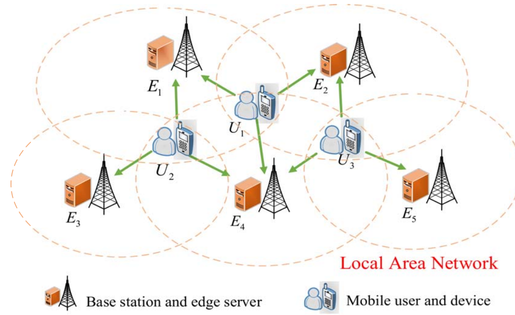


Fig. 1. Illustration of computing offloading. Users in the ellipse are accessible to the corresponding base stations, and user can offload a percentage of the current application to one of these stations.

and performance to determine whether the application should be executed locally or offloaded to the edge server. To take full advantage of the resources of mobile devices and edge servers, instead of only choosing either locally or remotely executing an application, we need to consider a partial offloading with a specific percentage. In [4] [5], the authors considered multi-frame object detection applications, and compared the performance of full and partial offloading. Experimental results showed partial offloading has a significant QoE (i.e. the delay or energy consumption) improvement.

In this paper, we re-examine the computation offloading problem. We consider a real-time multi-user multi-server offloading scenario, where the network and available resources of servers and clients are time-variant. As Figure 1 shows, the mobile base stations endowed with cloud-like computing and storage capability are densely distributed close to mobile users, and the user devices can offload computing task to the MEC servers through wireless channels. Indeed, the MEC server can be regarded as a substitute of the cloud [6] [7]. Users in the ellipse are accessible to the corresponding base stations, and user can offload a percentage of the current application to one of these stations. For example, user U_2 is in the coverage of base station E_1, E_3 and E_4. For each application at any slot t, the mobile device has to decide not only which server the application should be offloaded to, but also the proportion to be offloaded. The operation of mobile device can be viewed as a continuous decision-making process, which is likely to be solved by Reinforcement learning. Therefore,

978-1-7281-2583-1/19/$31.00 ©2019 IEEE
DOI 10.1109/ICPADS47876.2019.00066

we adopt a reinforcement learning method (i.e., ADQN) to solve this offloading problem, and experimental results show its feasibility and efficiency. To our best knowledge, we are the first to propose proportional computing offloading using DRL. Our main contributions are summarized as follows:

- We formalize the computation offloading problem as a long-term optimization problem, which aims to minimize the weighted sum of delay and energy consumption.
- We propose Advanced Deep Q Network algorithm (ADQN) to solve the offloading decision and proportion determination problem. We modify the DQN by adding Priority Replay Buffer and Expert Buffer Mechanism.
- We design a simulator to obtain enumerous training data to train ADQN. The results show that our proposed algorithm can significantly reduce the weighted sum of delay and energy consumption compared to other policies.

The remainder of this paper is organized as follows. Section II introduces the related work of computation offloading and deep reinforcement learning. Section III presents the system model and problem formulation. Section IV introduces the algorithm ADQN. Section V shows the experiment evaluation, and conclusions follow in VI.

II. Related Work

In this section, we review the related work on facets of computation offloading and machine learning-based schemes.

A. Facets of Computation Offloading

Computation offloading is the research focus in both mobile cloud computing (MCC) [6] [7] and mobile edge computing [1]–[3], which all concern how to make offloading decisions. Various works have studied different facets of this problem, e.g., energy efficient [8]–[11], resource allocation [12] [13], and some specific mobile applications such as AR/VR [5], [14]–[16]. In [11], the author adopted a big.LETTLE architecture, and aimed to minimize the energy consumption through better computation offloading policy. In [12], the author considered a multi-user mobile cloud computing system with a computing access point (CAP), where each mobile user has multiple independent tasks that may be processed at local/CAP/remote cloud. In [5], the author designed a framework that tied together front-end devices with more powerful backend "helpers" (e.g., home servers) to allow deep learning to be executed locally or remotely in the cloud/edge. In these above studies, the authors formulated the offloading problem as a NP-hard problem, and solved it through the heuristic algorithm. Nevertheless, these algorithms can not guarantee their robustness and can only be applied to specific scenarios, and thus they face a huge limitation.

B. Machine Learning-based Schemes

With the expansion of the Artificial Intelligence field, reseachers start attempting to solve the computation offloading problem by means of Machine Learning, and Reinforcement Learning (RL) [17] is the mainstream. Taking the future reward feedback from the environment into consideration, the RL agent can adjust its policy to achieve the best long-term goal. In [18], the author regarded computation offloading as a minority game, which consists of players (i.e., mobile devices), policy (i.e., locally or remotely executing tasks), and reward (i.e., QoE). Then, the Q-learning algorithm is adopted to solve this problem. In [19], the author designed a single-user task delay-optimal scheduling policy based on the theory of MDP, which controled the local processing, transmission units, and the task buffer queue delay. However, it is difficult to obtain the actual probability distribution of matrix P. In [20], the burgeoning DQN algorithm realized highly intelligent decision-making. In [21], the author considered a multi-user MEC system, and proposed RL-based optimization framework to tackle computation offloading and resource allocation for MEC. Nevertheless, the author only applied the original DQN without any modification or innovation. In [22] [23], the author adopted deep reinforcement learning to optimize the computation offloading performance in the virtual edge computing system, but the author just made assumptions instead of exploring facts in many details. Of course, the technology that applies reinforcement learning to edge computing is also maturing. In [24], the authors proposed adaptive video streaming with pensieve, which greatly optimized network links and improved service quality. In [25], the author achieved efficient management of the edge server with deep reinforcement learning. In [26], the authors proposed *Experience-driven Networking* based on DRL, which significantly reduces end-to-end delay and consistently improves the network utility.

Current studies focus more on coarse-grained offloading (i.e. full offloading or total offloading) [12] [18]. Due to its constrained capability to coordinate the local and server resources, the fine-grained offloading [8] [27] [28](i.e., partial offloading or dynamic offloading) needs further in-depth study. In view of these above problems, we are committed to designing a flawless Deep Reinforcement Learning model to solve the computation offloading problem in the remainder of this paper.

III. System Model and Problem Formulation

In this section, we first present our models, including the network model, application model, local execution model and remote execution model. Then, we present our long-term optimization problem.

A. Network Model

In the current MEC architecture, the edge servers are in charge of managing the resources and virtualizing the resources by means of VM, and the network deployment is based on the orthogonal frequency division multiple-access (OFDMA). We assume the total bandwidth B is divided into N subcarriers. The number of current available subcarriers is $k \in \mathcal{N} = \{1, 2, \cdots, N\}$. In order to better reflect the dynamic nature of the network condition, we divide the network links into upstream links and downstream links. Let p_u, p_s denote the transmission capacity for the mobile device and edge server, repectively. Assuming that the downlink and uplink transimissions experience the same noise, we can get the

TABLE I
NOTATIONS USED IN OUR FORMULATION.

h_{ul}, h_{dl}	the channel fading coefficient for uplink and downlink
N_0, B	the noise power and total bandwidth
g_{ul}, g_{dl}	the target BER for uplink and downlink
β_l	the path loss exponent
$\mathcal{B}, \mathcal{W}$	the total bytes of input data and total workload
ω	the number of clock cycles a microprocessor will perform per byte $cycles/byte(cpb)$
x_{ij}, y_{ij}	the index of the target server and target proportion
$P_{y_{ij}}$	the percentage of application A_{ij} that is offloaded to the edge server $E_{x_{ij}}$
$f_{ij}^{(l)}$	the local computing capability
e_l	the energy consumption per byte
$t_q^{(l)}$	the local queuing delay
$f_{x_{ij}}^{(r)}$	the computing capability of $E_{x_{ij}}$
e_r	the energy consumption per byte of servers
$t_q^{(r)}$	the remote queuing delay
λ, β	the weight factors to balance the delay and energy consumption
s, a	the current state and action
s', a'	the current state and action
s, a	the next state and action
α	the learning rate
γ	the incentive decay coefficient

maximum achievable rate (in bps) for uplink and downlink over an additive white Gaussian noise (AWGN) channel as

$$r_{ul} = k\frac{B}{N}\log\left(1+\frac{p_u\left|h_{ul}\right|^2}{\Gamma\left(g_{ul}\right)d^{\beta_l}N_0}\right), \tag{1}$$

and

$$r_{dl} = k\frac{B}{N}\log\left(1+\frac{p_s\left|h_{dl}\right|^2}{\Gamma\left(g_{dl}\right)d^{\beta_l}N_0}\right), \tag{2}$$

where d is the distance between the mobile device and server, N_0 is the noise power, β_l is path loss exponent, h_{ul}, h_{dl} are the channel fading coefficient for uplink and downlink, and g_{ul}, g_{dl} are the target BER (i.e., bit error rate) for uplink and downlink, respectively.

B. Application Model

The mobile device generates a compute-intensive application $\mathcal{G}$ at each time slot t. Based on [29] [30], the application $\mathcal{G}$ can be divided into multiple mutually independent tasks (i.e., a fine-grained partitioning) $\{c_1, c_2, \cdots, c_n\}$, and each component can be executed independently locally or offloaded to the edge servers. For example, a real-time video analytics application needs to analyze real-time images from different cameras at a single time. Because the pictures taken by different cameras are independent, we can divide this application into multiple independent tasks, and each task analyzes the pictures from the same camera.

We measure the workload of application according to the scale of input data. We denote the workload as $\mathcal{W}$, and total bytes of input data as $\mathcal{B}$. For a given $\mathcal{B}$, we can calculate $\mathcal{W} = \omega\mathcal{B}$, where ω in CPU $cycles/byte(cpb)$ indicates the number of clock cycles a microprocessor will perform per byte of data processed. The parameter ω depends on the nature of the application, e.g., the time and space complexity. Therefore, we can get the proportion of each task based on their bytes, namely $\{p_1, \cdots, p_n\}$, $\sum_{i=1}^{n} p_i = 1$.

C. Local and Remote Execution Model

Without loss of generality, we choose the delay and energy consumption as the main indicators. The delay includes computing delay, data transmission delay and queuing delay. As Figure 2 shows, we roughly divide application $\mathcal{G}$ into multiple independent tasks $\{c_1, c_2, \cdots, c_n\}$, and calculate the $\{p_1, \cdots, p_n\}$, $\sum_{i=1}^{n} p_i = 1$. Then, we divide these tasks into two parts, and calculate their proportion. We use $\{(P_1, Q_1), (P_2, Q_2), \cdots, (P_N, Q_N)\}$ to denote the percentage of workload executed at server or local device, where $P_i + Q_i = 1$. Assuming the user set is $\{U_1, U_2, \cdots, U_Z\}$, the edge server set is $\mathcal{M} = \{E_1, E_2, \cdots, E_M\}$, and time $\mathcal{T}$ is divided into slots $\{t_1, t_2, \cdots, t_s\}$. For any user U_i at each time slot t_j, the mobile device of U_i has to handle a different application A_{ij}. If no application A_{ij} is generated, we set $\mathcal{B}_{ij} = 0$. Let x_{ij} denote the index of the target server, and y_{ij} denote the index of target proportion, then $P_{y_{ij}}$ is the percentage of application A_{ij} that is offloaded to the edge server $E_{x_{ij}}$, where $x_{ij} \in \{1, 2, \cdots, M\}$, $y_{ij} \in \{1, 2, \cdots, N\}$. In this article, application execution consists of two parallel components, namely local exection and remote execution. We consider three metrics for each application, i.e. transmission delay, computing delay, and energy consumption.

1) Local execution: The local execution model is relatively simple, and we just consider local computing delay and energy consumption. Therefore, for user U_i at time slot t_j, the delay $D_{ij}^{(l)}$ and energy consumption $E_{ij}^{(l)}$ of local execution are

$$T_{ij}^{(l)} = D_{ij}^{(l)} + t_q^{(l)} = \frac{w\mathcal{B}_{ij}Q_{y_{ij}}}{f_{ij}^{(l)}} + t_q^{(l)}, \tag{3}$$

and

$$E_{ij}^{(l)} = e_l\mathcal{B}_{ij}Q_{y_{ij}}, \tag{4}$$

where $f_{ij}^{(l)}$ is the local computing capability, e_l is the energy consumption per byte, $t_q^{(l)}$ is the local queuing delay, $\mathcal{B}_{ij}$ is the size of input.

2) Remote execution: The remote execution model in the edge server is much more complex than that in the mobile device, because the data transmission dalay in the network must be taken into account, especially for applications with large data scale. In practice, the computing delay, computing energy consumption, and queuing delay in the servers are much smaller than those in mobile devices. Therefore, we can calculate the delay $D_{ij}^{(r)}$ and energy consumption $E_{ij}^{(r)}$ of local execution as

$$T_{ij}^{(r)} = D_{ij}^{(r)} + t_q^{(r)} = \frac{w\mathcal{B}_{ij}P_{y_{ij}}}{f_{ij}^{(r)}} + t_q^{(r)} + \frac{\mathcal{B}_{ij}P_{y_{ij}}}{r_{ul}^{(ij)}} + \frac{w^s\mathcal{B}_{ij}}{r_{dl}^{(ij)}}, \tag{5}$$

and

$$E_{ij}^{(r)} = e_r\mathcal{B}_{ij}P_{y_{ij}}, \tag{6}$$

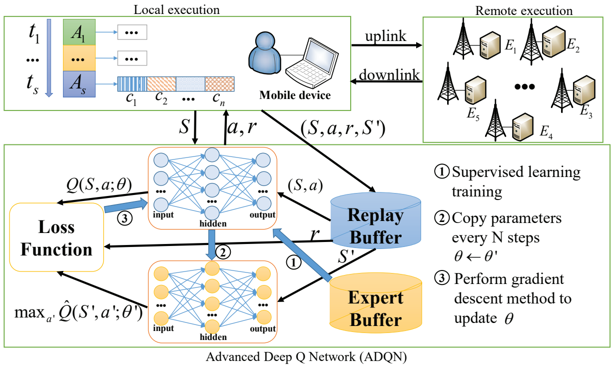


Fig. 2. System Overview. At each time slot, mobile device divides an application into several independent components and decides the target server and target proportion via ADQN, which continuously updates the network parameters by means of batch gradient descent.

where $f_{x_{ij}}^{(r)}$ is the computing capability of $E_{x_{ij}}$, e_r is the energy consumption per byte of servers, and $t_q^{(r)}$ is the remote queuing delay. ω^s is the discount coefficient, and $\omega^s \mathcal{B}_{ij}$ is the bytes of computed result. By synthesizing the above analysis of these two models, we can get the total weighted sum of delay and energy consumption

$$W_{ij} = \lambda \max\left(T_{ij}^{(r)}, T_{ij}^{(l)}\right) + \beta\left(E_{ij}^{(r)} + E_{ij}^{(l)}\right), \qquad (7)$$

where λ, β are the weight factors to balance the delay and energy consumption, λ, $\beta \in (0, 1)$. In practice, users prefer to pursue a better latency, and thus we are more inclined to set λ to a number bigger than β.

D. Problem Formulation

We have established an execution model for a single application, and now we focus on executing a series of applications as Figure 2 shows. Time $\mathcal{T}$ is divided into slots $\{t_1, t_2, \cdots, t_s\}$. For any user U_i at each time slot t_j, the mobile device of U_i has to handle a different application A_{ij}. We aim to find the optimal x_{ij} and y_{ij}. Therefore, the problem $\mathcal{P}_1$ is formulated as follows:

$$\begin{aligned} \mathcal{P}_1 : \min \sum_{t_j \in \mathcal{T}} & \left[\lambda \max\left(T_{ij}^{(r)}, T_{ij}^{(l)}\right) + \beta\left(E_{ij}^{(r)} + E_{ij}^{(l)}\right)\right] \\ s.t. \quad & f_{x_{ij}}^{(r)} \leq \mathcal{F}_{x_{ij}}^{(r)},\ f_{x_{ij}}^{(l)} \leq \mathcal{F}_i^{(l)} \\ & r_{ul}^{(ij)} \leq \mathcal{R}_{ul}^{(i)},\ r_{dl}^{(ij)} \leq \mathcal{R}_{dl}^{(i)} \\ & x_{ij} \in \{1, 2, \cdots, m\},\ y_{ij} \in \{1, 2, \cdots, n\}, \end{aligned} \qquad (8)$$

where $\mathcal{F}_{x_{ij}}^{(r)}$, $\mathcal{F}_i^{(l)}$ are the maximum available computing capability of $E_{x_{ij}}$ and U_i, $\mathcal{R}_{ul}^{(i)}$ and $\mathcal{R}_{dl}^{(i)}$ are the maximum available uplink and downlink capacity, respectively.

Apparently, problem $\mathcal{P}_1$ is an integer programming problem, which aims to find optimal values of x_{ij} and y_{ij}, and then gets the target server $E_{x_{ij}}$ and target proportion $P_{y_{ij}}$. However, it is a NP-hard problem extended from the Knapsack problem. Instead of solving this NP-hard problem by conventional optimization methods, we propose a reinforcement learning-based method to get the optimal $E_{x_{ij}}$ and $P_{y_{ij}}$ directly.

IV. Algorithm Design

In this section, we give a detailed elaboration on the reinforcement learning-based method. Firstly, we do some preliminaries about defining the specific three elements of Reinforcement Learning in the offloading scenario. Then, We briefly introduce the classical Q-learning theory. At last, we propose our Advanced Deep Q Network algorithm.

A. Preliminaries

In our scenario, we train an offloading policy for each device, thus the device can be viewed as RL-agent. Normally, if the exact state transition probability matrix $\mathcal{P}$ is obtained, we can solve the offloading problem perfectly with Markov Decision Process theory represented by a quaternion $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P})$. However, it is hard to get the transition probability because of the time-variant edge enviroment. Therefore, we consider model-free RL represented by a triple $(\mathcal{S}, \mathcal{A}, \mathcal{R})$.

- **State.** Reinforcement learning aims to constantly learn strategies from historical information to approach God's perspective, and thus a comprehensive definition of state is critical to decision-making efficiency. We take application, network condition, and available computing resources into consideration, and define the state at time slot t as $s_t = (\mathcal{B}, r_{ul}^1, \cdots, r_{ul}^m, r_{dl}^1, \cdots, r_{dl}^m, C_l, C_1, \cdots, C_m)_t$ where $\mathcal{B}$ represents the scale of the input data, and C_l is the available computing resources of the local device. C_i is the available computing resources of edge server i, r_{dl}^m and r_{ul}^m denote the downlink capacity and uplink capacity of server m, respectively.
- **Action.** In our offloading scenario, we view mobile device as an RL-agent, which will make a decision on the target server E_{tar} and target proportion P_b when receiving a state (or observation) s. Innovatively, we joint the offloading decision making and proportion determination, and define the action as a vector $a_t = (E_{tar}, P_b)_t$.
- **Reward.** At each time slot t, the agent will get a reward $R(s_t, a_t)$ in a certain state s_t after executing action a_t. In practice, the reward function should be positively correlated with the objective function. In section III, the objective is to minimize the weighted sum of delay and energy consumption, while the goal of RL is to maximize the long-term reward. Therefore, the reward function should be negatively related to the weighted sum of delay and energy consumption. We define the immediate reward as normalized $\frac{W_l - W(s,a)}{W_l}$, where W_l is the sum cost if the whole application is executed locally, and $W(s, a)$ is the sum cost by adopting RL methods. If $W(s, a) > W_l$, the reward is a negative number, which is more conductive for RL-agent to degrade bad actions.

Notice that, the mobile device has no prior knowledge of the real-time available bandwidth (i.e., r_{up}, r_{dl}) and cores of servers (i.e., C_i). Therefore, we propose a Server Broadcasting Mechanism. Each mobile device maintains an information table that records local available computing resources, available uplink and downlink capacity, and available computing resources of each server the user can access to. At each time slot t, each edge server will send *heartbeat package* to mobile devices in its domain, including the available computing resources and downlink capacity. Each user will modify the table based on the broadcast information. The model training process is performed on the server and the model is updated regularly. Each user will periodically download the latest model from the server, whereby the user can make efficient decisions based on the current model.

B. Advanced Deep Q Network algorithm

As one of the classic value-based RL algorithms, Q-learning has a critical impact on the successive study of RL, e.g., Deep Q Network (DQN). We advance the DQN and make it fit a more complex scenario.

1) Theory of Q-learning: The main idea of Q-learning is to construct a Q-table between State and Action to store Q value, and then select the action corresponding to the maximum Q value. Each state-action pair will have a value $Q(s,a)$, which can be regarded as a long-term reward. In the continuous training process, we can update Q value by means of TD-error until Q table converges. The method of updating Q value is

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q\left(s', a'\right) - Q(s,a) \right], \quad (9)$$

where s, a are the current state and action, s', a' are the next state and action, α is learning rate, and γ is incentive decay coefficient, respectively.

2) ADQN: By consulting the Q-table, the agent can make decisions quickly. However, the performance of Q-learning is not ideal in many cases. If the state and action space of the problem are very large, it is unrealistic to record all $Q(s,a)$ with Q table. What's worse, we have to calculate all the state-action values to ensure that for any state and action, we can get the corresponding value. Taking the Atari game as an example, after image preprocessing, the number of state is $256^{84\times 84}$ and the number of actions is 10, so the total number of state-action pairs is $256^{84\times 84} \times 10$. It is a challenge for the existing computer to store all the Q values.

In the offloading scenario, assuming that the maximum size of input date is $\mathcal{D}$, the link capacity fluctuates between x to y, the maximum CPU cores of local device is 8, and remote server 32, so the number of state spaces is $\mathcal{D} \times (y-x)^2 \times 8 \times 32^M$, without taking the number of action spaces into account. Therefore, we use a Deep Neural Network to estimate $Q(s,a)$ instead of computing Q value for each state-action pair, which is also the basic idea of Deep Q Network (DQN). Unfortunately, the original DQN has two big problems as follows:

- Each sample is selected to train the network with the same probability. In fact, the harvest of learning each sample varies with the difficulty of each sample. Simple samples do not improve the model much, but difficult samples do well. If we treat each sample equally, we will spend much time on simple samples, and the potential of learning can't be fully tapped.

Algorithm 1: Advanced DQN (ADQN)

Input: Initialized replay buffer D with expert data, Behaviour Network Q with θ, Target Network $\hat{Q}$ with θ'

Output: Q value, convergent θ and θ'

1 **for** $episode = 1, M$ **do**
2   Choose a random initial state s_1;
3   get preprocessed state $\phi_1 = \phi(s_1)$;
4   **for** *step* $s = 1, K$ **do**
5   Sampling random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D with expert data;

$$y_j = \begin{cases} r_j & \text{terminates} \\ r_j + \gamma \max_{a'} J(\hat{Q}(\phi_{j+1}, a'; \theta')) & \text{otherwise} \end{cases}$$

$$loss = (y_j - J(Q(\phi_j, a_j; \theta)))^2;$$

6   Perform a gradient descent to θ;
7   **if** $step \% n == 0$ **then**
8             $\theta' \leftarrow \theta$;
9   **for** $t = 1, T$ **do**
10   With probability ϵ select a random action a_t;
11   Otherwise select $a_t = \arg\max_a Q(\phi(s_t), a; \theta)$;
12   Execute action a_t in emulator;
13   Observe reward r_t and next state s_{t+1};
14   Preprocess $\phi_{t+1} = \phi(s_{t+1})$;
15   Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D;
16   **if** *D is saturated and* $t \% m = 0$ **then**
17   Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D with Priority Replay Buffer Mechanism;
18   The same as lines 5-8;

- Cold start problem. For the value-based RL algorithm such as Q-learning algorithm, it is difficult for the model to reach a relatively ideal state quickly in the early iterations. Moreover, the large deviation of estimating value function in early stage, and the apparent errors between environmental sampling and optimal strategy make learning much more difficult.

Priority Replay Buffer Mechanism [31] solved the first problem perfectly. It assigns each sample a weight based on the sample's performance or contribution to the model, and the probability of each sample to be selected is related with its weight. The less the contribution, the lower the weight, and the lower probability to be selected.

Now we focus on the second problem, and the main idea is to use pre-prepared high-quality samples to accelerate the training speed in the early stages of the model training. We call such samples expert samples. The high quality sampling trajectory comes from better strategies, and thus the model training is equivalent to standing on the shoulders of giants and the learning speed will naturally be much faster. As long as a certain number of high quality trajectories have been sampled, we can complete the pre-training through supervised learning before the agent interacts with the environment. In the offloading scenario, we can obtain expert data through some greedy policies. For each application, the mobile device selects the server with the maximum available computing resources or link capacity as the target server, and chooses an intermediate percentage (i.e., 50%) as the target proportion. Typically, the expert data perform well in early stages.

In addition, we need to use DQN to complete the training. The objective function of the ADQN becomes a combination of multiple learning objectives:

$$J(Q) = J_{DQN}(Q) + \lambda_1 J_E(Q) + \lambda_2 J_{L2}(Q), \quad (10)$$

where $J_{DQN}(Q)$ is the objective function of $Q(s,a)$, $J_E(Q)$ is the objective function of supervised learning, $J_{L2}(Q)$ is the objective funxtion of $L2-Regularization$,and λ_1, λ_2 are the weight factors to balance these objective functions.

In general, the expert samples are limited, so it can only affect a small part of the state-action value. Moreover, if these data cannot cover as many states as possible, they may have a negative impact on the model. To avoid these problems, we define the objective function of supervised learning as follows:

$$J_E(Q) = \max_{a \in \mathcal{A}}[Q(s,a) + l(a_E, a)] - Q(s, a_E), \quad (11)$$

where a_E is the action given by experts, $l(x, y)$ is an indicator function. If $a = a_E$, then $l(a, a_E) = 0$, and $J_E(Q) = 0$, which means that the decision-making of the model is consistent with that of experts. If $a \neq a_E$, it shows that the value of one other action is not much worse than that of expert actions.

C. Summary

Now, we present the overview of computation offloading with our proposed ADQN algorithm as shown in Figure 2. For application $\mathcal{G}$ at each slot t, the mobile device divides the application into multiple independent tasks, and each task corresponds to a proportion. Then, the client has to make an offloading decision on the target server and target proportion. We define $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{R}$ in our proposed ADQN algorithm. The model training process is performed in edge server, and users regularly download the latest models from the server. The client runs the model with the current state $\mathcal{S}$ as input to get the Q value of all actions, e.g., $\{Q(\mathcal{S}, a_1), Q(\mathcal{S}, a_2), \cdots, Q(\mathcal{S}, a_n)\}$ and selects the action corresponding to the maximum Q value $a' = \arg\max_{a \in \mathcal{A}} Q(\mathcal{S}, a)$. Based on this action $a' = (E_{tar}, P_b)$, the client offloads the application to the target server E_{tar} with target proportion P_b.

To sum up, we present ADQN algorithm as Algorithm 1. First of all, we initialize replay buffer D with expert data, Behaviour Network Q with θ, and Target Network $\hat{Q}$ with θ'. Then, we get the relative ideal weight θ_1 and θ'_1 through supervised learning with expert data. After that, we use the behavior network to interact with RL-agent (i.e., mobile device) to get a series of samples, and store them in the replay buffer. When the samples reach a specified number, every several steps we randomly sample a minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D with Priority Replay Buffer Mechanism to train the network, until the two networks are convergent.

TABLE II
PARAMETER SETUP.

Parameter	Value	Parameter	Value	Parameter	Value
p_u	$0.01W$	p_s	$0.1W$	β_2	0.1
g_{ul}	10^{-3}	g_{dl}	10^{-3}	λ_1	0.9
B	3	β_l	-2	λ	0.6
d	10	N	256	λ_2	0.4

V. EXPERIMENT EVALUATION

In this section, we evaluate the performance of our proposed ADQN algorithm. In the evaluation, based on the real-time video analysis application, we design our simulation experiment, and assign specific value to the parameter. Then, we analyze the feasibility of ADQN. At last, we design several control groups to verify the efficiency of the ADQN algorithm.

A. Parameter Setup

Our simulation experiment can be migrated from a real scene as follows. In a real-time video analysis application such as feature detection, the terminal device may receive photos from different cameras at the same time. When the available computing resources of the terminal device are insufficient, a certain proportion of the photos need to be offloaded to the appropriate edge server for execution. Therefore, the terminal device has to decide the target edge server and proportion.

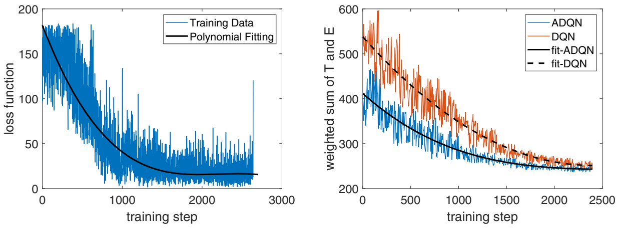
Assuming that each terminal device executes the applications between 8 a.m. and 5 p.m., and application is generated every five minutes. From 8:00 a.m. to 11:00 a.m., due to the working hours, the size of the application will gradually increase, the available local and server computing resources, and the available link capacity will decrease. From 11:00 a.m. to 2:00 p.m., due to the break time, the volatility trend is contrary to that in the morning. From 2:00 p.m., the changing trend of various elements is the same as in the morning.

In general, we can set up our parameters based on the above scene. The size of each initial application $\mathcal{B}$ follows an Uniform Distribution, $\mathcal{B} \in [2000, 3000]$. The number of initial available subcarriers k and computing resources (i.e., cores) f_s and f_l follow an Uniform Distribution, $k \in [160, 200]$, $f_s \in [25, 32]$, $f_l \in [6, 8]$, respectively. At each slot t, the variations of $\Delta\mathcal{B}$, Δk, Δf_l, Δf_s follow a Poisson Distribution with parameter $\mathcal{P}$ = 30, 10, 2, and 5, respectively. Some other constant parameters are shown in table II. Furthermore, we need to define various model parameters for our training, such as the episodes K, the Replay Buffer size D, the Mini-batch size m, the learning rate α, the Reward decay γ, the network weight update frequency f, and greed index ϵ. In the following comparative experiments, we will analyze the effects of these parameters on the experimental results.

B. Comparison Design

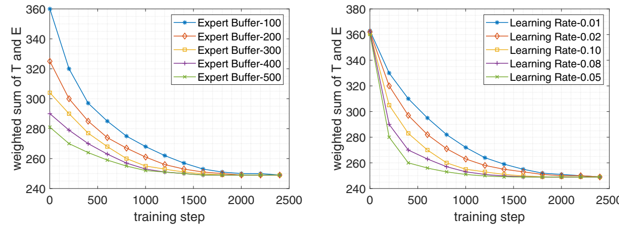
To verify the performance of ADQN, we use several strategies as benchmarks:

- None Offloading Policy (NOP). All the components of the application are exexuted locally, and thus NOP doesn't care about link capacity and server resources.



(a) Convergence (b) ADQN vs DQN

Fig. 3. Convergence and superiority of ADQN. Take note that denotation of Y-axis in (a) and (b) is different, which denotes the training cost in (a), and the weighted sum of delay (T) and energy consumption (E).



(a) Effect of various expert buffer size (b) Effect of various learning rate

Fig. 4. Effect of learning rate and expert buffer size on ADQN performance. Y-axis denote the weighted sum of delay (T) and energy consumption (E).

- Random Offloading Policy (ROP). At each slot t, the mobile device randomly chooses an action $a \in \mathcal{A}$, including random server and random proportion.
- Link Capacity Optimal Policy (LCOP). At each slot t, by querying the server resource table, the mobile device chooses the server with the largest available link capacity as the target server, and offloads the whole application to the target server.
- Computing capability Optimal Policy (CCOP). Similar to LCOP, the mobile device pursues the optimal available computing resources. Therefore, the whole application will be offloaded to the target server with the largest available computing resources.

C. Simulation Results

Firstly, we show the feasibility of ADQN. Since we estimate the Q value by means of deep neural network, the network is guaranteed to be convergent. Otherwise, the inaccurate estimation will significantly affect the decision-making performance in the later stages. We set K to 300, D to 2000, α to 0.01, m to 50, γ to 0.9, f to 200, and ϵ to 0.9. Then, we start training the network and get Figure 5. As Figure 3(a) shows, it is clear that with the increase of training steps, the training cost gradually decreases, and eventually goes to zero, i.e. convergence, which shows that it is feasible to approximate Q value with neural network. The network is constantly fluctuating because RL-agent selects action with probability in each step, so when the action is poor (resp. good), it will get a low (resp. high) reward. We can also use a polynomial expression to fit the training data. Therefore, it is feasible to solve the offloading problem with ADQN. In addition, to verify the efficiency of ADQN over DQN, we designed a comparative experiment as Figure 3(b) shows. Every 10 training steps, we use the current ADQN and DQN model for decision-making, and calculate the weighted sum of delay and energy consumption. Although there are some cases DQN outperforms ADQN, ADNQ has a significant improvement compared with DQN at early stages of training due to the Priority Replay Buffer Mechanism and Expert Buffer Mechanism. With the increase of training steps, ADQN and DQN will converge gradually, and eventually achieve almost the same effect.

Secondly, we analyze the effect of buffer size and learning rate on ADQN performance. As Figure 4(a) shows, with the increase of expert buffer size, ADQN can be capable of a more ideal performance at the initial stage. However, the performance improvement is not linear with the expert buffer size. When a certain number is reached, the performance improvement is not significant with the increase of expert buffer size. Thus it is critical to determine the size due to the costly expert buffer. Figure 4(b) shows that learning rate plays a decisive role in the convergence of the model. However, a high learning rate does not necessarily accelerate the convergence rate. For example, ADQN with the learning rate $\alpha = 0.08$ or 0.10 converges more slowly than $\alpha = 0.05$.

Finally, we compared the performance of ADQN with other strategies. Similar to the above method, we set K to 300, D to 2000, α to 0.01, m to 50, γ to 0.9, f to 200, and ϵ to 0.9. Every 200 training steps, we use the current ADQN model and other offloading policies for decision-making, and calculate the weighted sum of time delay and energy consumption. The result is shown in Figure 5. In early stages of training, ADQN is only a little better than NOP and ROP, and much worse than the other two greedy strategies (i.e., LCOP and CCOP), which is largely due to the strong randomness of action selection. With the increase of training steps, ADQN gradually increases the probability of optimal action, and makes each decision closer to the optimal choice. LCOP and CCOP are unilaterally pursuing the maximization of resources (i.e., subcarriers k or cores f), which is a good method in many cases, but it is too one-sided to consider. For example, when the network is congested, CCOP will perform poorly. Therefore, the overall performance is slightly inferior to ADQN.

VI. Conclusion

In this paper, we rethink the computation offloading problem, and propose Advanced Deep Q Network algorithm to solve it. Firstly, we consider a general real-time multi-user, multi-server offloading scenario, in which the edge server is co-located with base station, and users release applications irregularly. For each application produced by users, we aim to minimize its weighted sum of delay and energy consumption by offloading an ideal proportion of workload to an optimal edge server. Then, we formalize the offloading problem as a long-term optimization problem. Considering the complexity of solving this *NP-hard* problem, we make full use of the time-variant edge enviroment, and adopt modified DQN by adding Priority Replay Buffer Mechanism and Expert Buffer Mechanism. The experimental results show that ADQN has strong practicability and efficiency. Generally, it is unrealistic

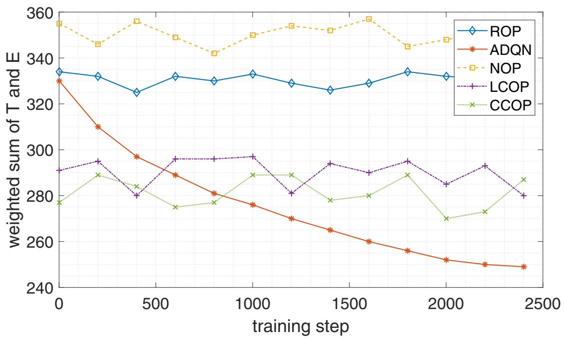


Fig. 5. Performance comparison between ADQN and other strategies.

to train the RL network in a real scenario for constant trail and error. Therefore, go a step further, we are going to design a simulator that is extremely consistent with the real world.

Acknowledgments

This work was supported in part by National Key R&D Program of China (2017YFB1001801), NSFC (61872175, 61832008), Natural Science Foundation of Jiangsu Province (BK20181252), Jiangsu Key R&D Program (BE2018116), and Collaborative Innovation Center of Novel Software Technology and Industrialization. Sheng Zhang is the corresponding author.

References

[1] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "Mobile edge computing: Survey and research outlook," *arXiv preprint arXiv:1701.01090*, 2017.

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

[3] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing: A key technology towards 5g," *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.

[4] Z. Lu, K. S. Chan, and T. L. Porta, "A computing platform for video crowdprocessing using deep learning," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, April 2018, pp. 1430–1438.

[5] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "Deepdecision: A mobile deep learning framework for edge video analytics," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, April 2018, pp. 1421–1429.

[6] N. Fernando, S. W. Loke, and W. Rahayu, "Mobile cloud computing: A survey," *Future generation computer systems*, vol. 29, no. 1, pp. 84–106, 2013.

[7] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation computer systems*, vol. 25, no. 6, pp. 599–616, 2009.

[8] J. Xu, L. Chen, and S. Ren, "Online learning for offloading and autoscaling in energy harvesting mobile edge computing," *IEEE Transactions on Cognitive Communications and Networking*, vol. 3, no. 3, pp. 361–373, Sep. 2017.

[9] Y. Sun, S. Zhou, and J. Xu, "Emm: Energy-aware mobility management for mobile edge computing in ultra dense networks," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2637–2646, Nov 2017.

[10] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, April 2018, pp. 207–215.

[11] Y. Geng, Y. Yang, and G. Cao, "Energy-efficient computation offloading for multicore-based mobile devices," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, April 2018, pp. 46–54.

[12] M. Chen, B. Liang, and M. Dong, "Joint offloading and resource allocation for computation and communication in mobile cloud with computing access point," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, May 2017, pp. 1–9.

[13] S. Barbarossa, S. Sardellitti, and P. Di Lorenzo, "Joint allocation of computation and communication resources in multiuser mobile cloud computing," in *2013 IEEE 14th Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, June 2013, pp. 26–30.

[14] R. D. Yates, M. Tavan, Y. Hu, and D. Raychaudhuri, "Timely cloud gaming," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, May 2017, pp. 1–9.

[15] L. Wang, L. Jiao, T. He, J. Li, and M. Muhlhauser, "Service entity placement for social virtual reality applications in edge computing," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, April 2018, pp. 468–476.

[16] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman, "Live video analytics at scale with approximation and delay-tolerance," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, Mar. 2017, pp. 377–392.

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.

[18] S. Ranadheera, S. Maghsudi, and E. Hossain, "Computation offloading and activation of mobile edge computing servers: A minority game," *IEEE Wireless Communications Letters*, vol. 7, no. 5, pp. 688–691, Oct 2018.

[19] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *2016 IEEE International Symposium on Information Theory (ISIT)*, July 2016, pp. 1451–1455.

[20] 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, 2015.

[21] J. Li, H. Gao, T. Lv, and Y. Lu, "Deep reinforcement learning based computation offloading and resource allocation for mec," in *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, April 2018, pp. 1–6.

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

[23] 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 of Things Journal*, vol. 6, no. 3, pp. 4005–4018, June 2019.

[24] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: ACM, 2017, pp. 197–210.

[25] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, ser. HotNets '16. New York, NY, USA: ACM, 2016, pp. 50–56.

[26] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, "Experience-driven networking: A deep reinforcement learning based approach," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, April 2018, pp. 1871–1879.

[27] Y. Zhang, D. Niyato, and P. Wang, "Offloading in mobile cloudlet systems with intermittent connectivity," *IEEE Transactions on Mobile Computing*, vol. 14, no. 12, pp. 2516–2529, Dec 2015.

[28] D. Huang, P. Wang, and D. Niyato, "A dynamic offloading algorithm for mobile computing," *IEEE Transactions on Wireless Communications*, vol. 11, no. 6, pp. 1991–1995, June 2012.

[29] Y. Kao, B. Krishnamachari, M. Ra, and F. Bai, "Hermes: Latency optimal task assignment for resource-constrained mobile computing," *IEEE Transactions on Mobile Computing*, vol. 16, no. 11, pp. 3056–3069, Nov 2017.

[30] M. Chen, B. Liang, and M. Dong, "Joint offloading decision and resource allocation for multi-user multi-task mobile cloud," in *2016 IEEE International Conference on Communications (ICC)*, May 2016, pp. 1–6.

[31] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2015.