Wireless Networks
https://doi.org/10.1007/s11276-021-02789-7

Energy-efficient UAV-enabled computation offloading for industrial internet of things: a deep reinforcement learning approach

Shuo Shi[1,3] · Meng Wang[1] · Shushi Gu[2,3] · Zhong Zheng[4]

Accepted: 20 August 2021

Abstract
Industrial Internet of Things (IIoT) has been envisioned as a killer application of 5G and beyond. However, due to the shortness of computation ablility and batery capacity, it is challenging for IIoT devices to process latency-sensitive and resource-sensitive tasks. Moblie Edge Computing (MEC), as a promising paradigm for handling tasks with high quality of service (QoS) requirement and for energy-constrained IIoT devices, allows IIoT devices to offload their tasks to MEC servers, which can significantly reduce the task process delay and energy consumptions. However, the deployment of the MEC servers rely heavily on communication infrastructure, which greatly reduce the flexibility. Toward this end, in this paper, we consider multiple Unmanned Aerial Vehicles (UAV) eqqipped with transceivers as aerial MEC servers to provide IIoT devices computation offloading opportunities due to their high controbility. IIoT devices can choose to offload the tasks to UAVs through air-ground links, or to offload the tasks to the remote cloud center through ground cellular network, or to process the tasks locally. We formulate the multi-UAV-Enabled computation offloading problem as a mixed integer non-linear programming (MINLP) problem and prove its NP-hardness. To obtain the energy-efficient and low complexity solution, we propose an intelligent computation offloading algorithm called multi-agent deep Q-learning with stochastic prioritized replay (MDSPR). Numerical results show that the proposed MDSPR converges fast and outperforms the benchmark algorithms, including random method, deep Q-learning method and double deep Q-learning method in terms of energy efficiency and task successful rate.

Keywords Energy efficiency · Deep reinforcement learning · Computation offloading · Mobile edge computing · Unmanned aerial vehicles

1 Introduction

Industrial Internet of things (IIoT), as an emergying communication paradigm, is expected to evolutionize manufacturing and also drive growth in productivity across various types of applications such as smart factories and smart grids [1]. It is important to provide the required quality of service (QoS) in terms of reliability and real-time to IIoT applications [2]. However, the battery capacity and processing ability of IIoT devices are limited, which imposes great challenges when handling tasks with characteristics of computation-sensitive and latency-sensitive.

This paper was presented in part at the EAI AICON 2020: 2nd EAI International Conference on Artificial Intelligence for Communications and Networks, December 19-20, 2020, Cyberspace. Compared with the conference paper, we have made the abstract more precisely. To make the introduction more susbstantial, we refer to more related papers to do a in-depth survey on recent works. The contributions of the paper are more siginificant. Furthermore, the detailed algorithm is given, and the simulation results are added to further prove the effectiveness of the proposed scheme. Finally, we propose a new discussion chapter to study the potential applications of our proposed algorithm in the field of wireless communication.

✉ Shuo Shi
crcss@hit.edu.cn

[1] School of Electronics and Information Engineering, Harbin Institute of Technology, Harbin 150001, China

[2] School of Electronics and Information Engineering, Harbin Institute of Technology (Shenzhen), Shenzhen 518055, China

[3] Peng Cheng Laboratory, Shenzhen 518055, China

[4] International Innovation Institute of HIT in Huizhou, Huizhou 516000, Guangdong, China

Published online: 12 October 2021

Recently, with the rapid developement of mobile edge computing in IIoT, local devices can choose to offload the latency-sensitive tasks to edge servers through wireless access which can effectively alleviate the computation stress in local [3]. Due to the high flexibility and controllbility, unmanned aerial vehicles (UAVs) equipped with computation capabilities can act the role of edge servers to enable edge computing even in the absence of wireless infrastructures [4]. Furthermore, the air-ground communication links between devices and UAVs are largely line of sight (LoS) links [15], which will improve the reliability in both uplink and downlink transmission. The studies of energy-efficient computation offloading in IIoT with UAVs assisted are still remain as an open issue. In [4], a single UAV servers as a moving cloudlet for mobile users, aiming to minimize the total energy consumptions by jointly optimizing the bits allocation for uplink and downlink communications. [5] considers a stochastic task arrival model and obtains an energy-efficient offloading solution through Lyapunov optimization. [6] considers a multi-UAV scenario and solving the non-convex computation offloading problem with iterative optimization algorithm. [7] studies the computation rate maximization problem under both partial and binary computation offloading modes subject to the energy constraints. [21] proposes an energy-efficient unmanned aerial vehicle (UAV)-enabled MEC framework, where multiple UAVs are deployed as edge servers to provide computation assistance to terrestrial users and NOMA is adopted to reduce the energy consumption of task offloading.

However, the solutions to the complex computation offloading problems in the aforementioned works [5–7] are based on one-shot optimization and fail to maximize the long-term performance of computation offloading. Moreover, these solutions are obtained under a specific model and require a priori knowledge of the network model, if the channel fading is fast, they cannot meet the requirement of real-time decision. Furthermore, the computation capabilities in devices are limited in IIoT, algorithms with high time complexity cannot run efficiently in the processor of devices. To obtain long-term rewards and real-time offloading decisions, deep reinforcement learning (DRL), as an emerging artificial intelligence technique, has been introduced to complex computation offloading problems with low complexity [8]. [9] proposes a deep dynamic scheduling algorithm to solve the offloading decisions problem with minimum energy costs using deep Q-learning (DQN). [10] considers a computation offloading problem with stochastic vehicle traffic, dynamic computation requests and time-varying communication conditions, and uses Q-learning as offline solution, DQN as online solution. [11] uses double DQN (DDQN) instead of DQN to learn the optimal computation offloading policy without knowing a priori knowledge of network dynamics. In [12], authors proposes a modified version of deep deterministic policy gradient (DDPG) to accelerate the converge speed of the computation offloading problem. [22] investigates a multi-dimensional resource management for unmanned aerial vehicles assisted vehicular networks, and formulate the resource allocation problem as a distributive optimization problem and solve it through a multi-agent DDPG method.

Motivated by the aforementioned issues, in this paper, we focus on the energy-efficient binary computation offloading problem in IIoT with UAV assited. IIoT devices can choose to execute the tasks locally or offload the tasks to UAVs through LoS links or offload the tasks to the remote cloud center through base stations. Compared with [4, 5, 7] which focus only on a single UAV, we use multiple moving UAVs as edge servers to provide computation offload opportunities, which is more practical due to the limited battery capacity of UAVs.Compared with [8–11], we introduce the low-comlexity prioritized replay method to train the neural network efficiently, which can accelerate the whole learning process.

We first propose the multi-UAV-enabled network model and formulate the computation offloading problem as a mixed integer non-linear programming (MINLP) problem. Branch and bound (BB) method can be used to solve the problem [13] but with prohibitively high computational complexity. [14] proposes an iterative optimization method to solve the MINLP problem, however, the complexity grows rapidly as the size of the network increases. To address this, we propose our multi-agent deep Q-learning with stochastic prioritized replay (MDSPR) to solve the problem distributedly and efficiently. Compared with [9–11] using normal DQN and DDQN, our method choose training experiences with high priorities instead of choosing randomly, which can accelerate the training process and improve convergence stability. The main contributions of this paper can be summarized as follows.

(1) We present a network model where multiple UAVs serve as edge servers offering computation offloading opportunities for IIoT devices. Considering the dynamic channel states and task property, we formulate the multi-UAV-enabled computation offloading problem as a MINLP problem aiming to minimize the total energy consumptions of IIoT devices while satsifying the QoS requirements.
(2) To solve the problem effectively with low complexity, we propose our MDSPR solution. We first formulate the computation offloading problem as a markov decision process (MDP). Then, by leveraging DQN, each IIoT device is able to choose action with the highest action-state value, which leads to

maximum long-time reward. Furthermore, we apply stochastic prioritised replay to accelerate the training time and improve the stability of convergence.

(3) We compare our MDSPR with normal DQN, DDQN, and other benchmark algorithms. Simulation results show that the MDSPR outperforms the benchmark algorithms in terms of energy-efficiency and task successful ratio. Compared with DQN and DDQN, MDSPR converges faster and has lower errors between target value and estimation value.

The rest of the paper is organised as follows. The system model and problem formulation are introduced in Sect. 2. Section 3 gives a detailed description of our proposed MDSPR. We present our simulation results in Sect. 4. Finally, we give the conclusions in Sect. 5.

2 System model and problem formulation

2.1 Network model

We consider a three-layer UAV-Enabled IIoT computation offloading network model where IIoT devices process the tasks assigned from IIoT control centers, each IIoT device can choose to execute the tasks locally or offload the tasks to the UAVs or offload the tasks to the remote cloud center through terristrial base stations, and send the execution results back to control centers, as shown in Fig. 1. UAVs with communication and computation abilities act the role of MEC servers. $\mathcal{M} = \{1, 2, \ldots, M\}$, $\mathcal{N} = \{1, 2, \ldots, N\}$ denote the devices' set and UAVs' set, respectively. IIoT devices are distributed in an area with a radius of K and UAVs hover above the devices with a height of H. The locations of devices and UAVs during a single time interval $t \in \{1, 2, \ldots, T\}$ can be expressed as $\mathbf{q}_i(t) = (x_i(t), y_i(t)), i \in \mathcal{M}$ and $\mathbf{p}_j(t) = (X_j(t), Y_j(t), H), j \in \mathcal{N}$, respectively. Moreover, tasks can be modelled as $W_i(t) = (s_i(t), c_i(t), d_i(t))$ according to [3], where $s_i(t)$ denotes the total bits of the task, $c_i(t)$ denotes the total CPU circles required by the task and $d_i(t)$ denotes the tolerant delay of the task. In this paper, we assume UAVs flying with a random speed v in the fixed area, $\mathbf{q}_i(t)$, $\mathbf{p}_j(t)$ remain unchanged in a single time slot and there is a coming task for each IIoT device in a single time interval.

2.2 Communication model

According to [15] the air-ground channel gain between ith IIoT device and jth UAV can be modelled as probabilistic LoS channel:

$$P_{LoS}(\theta_i^j) = \frac{1}{1 + a\exp(-b(\theta_i^j - a))}, \tag{1}$$

where θ_i^j is the elevation angle between ith IIoT device and jth UAV, a, b are the environment related parameters, the probability of NLoS channle is $P_{NLoS}(\theta) = 1 - P_{LoS}(\theta)$. The pathloss of LoS and NLoS channel can be denoted as:

$$PL_{LoS,i}^j(t) = \beta_0(\sqrt{H^2 + \left\|\mathbf{q_i}(t) - \mathbf{p_j}(t)\right\|^2})^{-\alpha}\eta_{LoS} \tag{2}$$

$$PL_{NLoS,i}^j(t) = \beta_0(\sqrt{H^2 + \left\|\mathbf{q_i}(t) - \mathbf{p_j}(t)\right\|^2})^{-\alpha}\eta_{NLoS} \tag{3}$$

$$h_i^j(t) = P_{LOS} \times PL_{LoS} + P_{NLoS} \times PL_{NLoS}, \tag{4}$$

where β_0 denotes the channel gain at the reference distance $d_0 = 1$ m, α is the attenuation coefficient. Tasks can be offloaded to UAVs or the cloud center through wireless

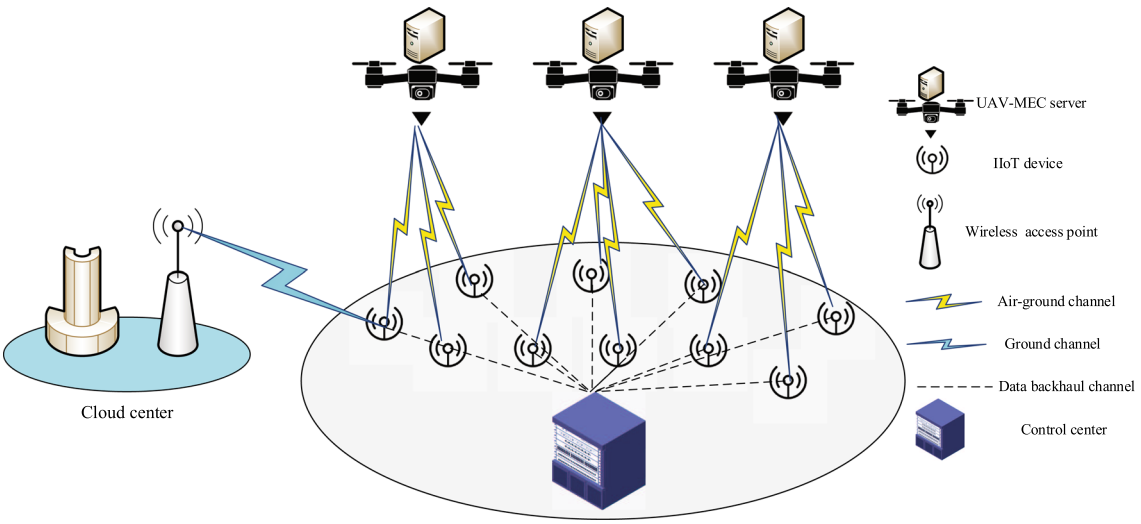

Fig. 1 Multi UAV-Enabled IIoT devices computation offloading network model

channle. Each IIoT device is associated with UAVs using orthogonal frequency-division multiplexing access (OFDMA). Tasks can be successfully offloaded to UAVs if the distance d between UAVs and devices is less than R, where R denotes the communication range of UAVs.

Since the size of the processed tasks can be negligible compared with the original tasks, in this paper, we only consider the uplink transmission. The uplink transmission rate between ith IIoT device and jth UAV can be expressed as

$$r_i^j(t) = B\log_2\left(1 + \frac{Ph_i^j(t)}{\sigma^2}\right), \tag{5}$$

where B is the allocated bandwith for IIoT devices, P is the uplink transmission power of IIoT devices, σ^2 denotes the background noise power, and we assume that it is constant on the slow fading channel [16]. The uplink transmission rate of an IIoT device i that chooses to offload the task to the cloud through wireless link can be denoted as

$$r_i^c(t) = \omega\log_2\left(1 + \frac{P_c H_i(t)}{\sigma^2}\right), \tag{6}$$

where w, P_c, $H_i(t)$ denote the bandwith, transmission power , the channel gain between ith IIoT device and cloud, respectively. For UAV computation offloading, the task transmission time can be expressed as

$$L_i^j(t) = \frac{s_i(t)}{r_i^j(t)}, \tag{7}$$

for cloud computation offloading, according to [17] the transmission time can be denoted as

$$L_i^c(t) = \frac{s_i(t)}{r_i^c(t)} + \tau, \tag{8}$$

where τ denotes the uplink propogation delay factor due to congestion of the bachhaul link.

2.3 Computation model

To simplify the analysis, all the UAVs are assumed to have the same computation capacity, denoted as C_u. The computatioin capacity of IIoT devices and the remote cloud center are denoted as C_l and C_c, respectively. Thus the local task computation time can be expressed as

$$L_l(t) = \frac{c_i(t)}{C_l}, \tag{9}$$

according to [17], the energy consumption of local computation can be given as

$$E_i^l(t) = \theta c_i(t)^2, \tag{10}$$

where θ is the factor denoting the consuming energy per CPU cycle. The task computation time in UAV can be denoted as

$$D_u(t) = \frac{c_i(t)}{C_u}. \tag{11}$$

Note that,the cloud processing time can be negligible since the computation capacity is enormous in cloud center. Thus, for UAV offloading cases, the total communication energy can be expressed as

$$E_i^j(t) = P(L_i^j(t) + D_u(t)), \tag{12}$$

for cloud offloading cases, the total energy consumptions can be modelled as

$$E_i^c(t) = P_c(L_i^c(t) + \tau), \tag{13}$$

2.4 Problem formulation

In each time interval IIoT devices should make offloading decisions, we define $\mathbf{I} = (I_1(t), I_2(t), \ldots, I_M(t))$ as the offloading indicated vector in a single time interval, where $I_i(t) \in \{I_i^{-1}, I_i^0, I_i^1, \ldots, I_i^M\}, \forall t \in T, \forall i \in \mathcal{M}$ is an indicator which can be used to describe an offloading decision, $I_i^k \in \{0, 1\}, k \in \{-1, 0, 1, 2, \ldots, N\}$. Let $E(t) \in \{E_i^c(t), E_i^j(t), E_i^l(t)\}$, $L(t) \in \{L_i^c(t), L_i^j(t), L_i^l(t)\}$. The energy consumption of a singe task can then be derived as

$$E_i(t) = I_i(t)E(t) = \begin{cases} E_i^c(t), & k = -1 \\ E_i^l(t), & k = 0 \\ E_i^j(t), & k \in \mathcal{M} \end{cases} \tag{14}$$

Our objective is to minimize the total energy cosumptions of all IIoT devices in a time period T through optimizing the offloading indicated vector while satisfying the basic constraints. Thus, we can formulate the energy-efficient multi-agent computation offloading problem as follows

$$\begin{aligned} \min_{\boldsymbol{I},\boldsymbol{p},\boldsymbol{q}} \quad & \sum_{t=1}^{T}\sum_{i=1}^{M} I_i(t)E_i(t). \\ \text{s.t.} \quad & C1: I_i^k \in \{0,1\}, \forall i \in \mathcal{M}, k \in \{-1,0,1,2,...,N\} \\ & C2: \sum_{k=1}^{N} I_i^k \le 1, \forall i \in \mathcal{M}, k \in \{-1,0,1,2,...,N\} \\ & C3: \sum_{i=1}^{M} I_i(t) \le W, \quad I_i(t) \in \{I_i^1, ..., I_i^M\} \\ & C4: I_i(t)L(t) \le d_i(t), \forall i \in \mathcal{M}, \forall t \in \mathcal{T} \\ & C5: H^2 + ||\mathbf{q}_i(t) - \mathbf{p}_j(t)||^2 \le R^2, \forall i \in \mathcal{M}, \forall t \in \mathcal{T} \end{aligned} \tag{15}$$

C1 indicates that I_i^k is a binary variable. C2 shows that if a task is choosen to offload to UAVs, it can only be offloaded to a single UAV. C3 gives the constraints that tasks need to be executed in UAVs cannot exceed the maximum limit. C4 guarantees that the task can be finished within the tolerant delay. C5 means if ith IIoT devices offload the task to jth UAV, the distance between the device and UAV should less than the communication range of the UAV.

Theorem 1 *The multi UAV-enabled computation offloading problem is NP-hard, more specifically, it is a mixed integer non-convex non-linear programming (MINLP) problem.*

Proof To show the NP-hardness, we first introduce the classic maximum cardinality bin packing problem. Given Y items with size q_i for $i \in \mathcal{Y}$ and G bins of identical capacity C, the objective is to assign a maximum number of items to the fixed number of bins without exceeding the capacity. The problem is formulated as

$$\begin{aligned} &\max \sum_{i=1}^{Y}\sum_{j=1}^{G} x_{i,j}, \\ &\text{s.t.} \sum_{j=1}^{G} x_{i,j} \le 1, \quad \forall i \in \mathcal{Y}, \\ &\sum_{i=1}^{Y} q_i x_{i,j} \le C, \quad \forall j \in \mathcal{G}, \\ &x_{i,j} \in \{0,1\}, \quad \forall i \in \mathcal{Y}, \forall j \in \mathcal{G} \end{aligned} \tag{16}$$

(6) has been proved as NP-hard in [18]. Note that, in our problem, C1, C2, C3 are exactly the constraints of (6). The objective function is equal to $\max_{\boldsymbol{I}} \sum_{t=1}^{T}\sum_{i=1}^{M} I_{min}$, where I_{min} denotes the offloading decision that costs minimum energy consumptions. Therefore, our optimization problem is a complex form of (6), which proves the NP-hardness. Since I_i^k is a binary variable, according to convex optimization theory, the problem is non-convex. Moreover, $\boldsymbol{p}$,$\boldsymbol{q}$ are both continious variable with C5 a quadratic constraint, so the problem is an MINLP problem, which completes the proof. □

Due to the NP-hardness and high complexity of MINLP, rarely methods can be applied directly to solve the energy-efficient multi-agent computation offloading problem currently. Exhaustive method and branch and bound (BB) algorithm can obtain an optimized solution but as the network size grows, the time complexity increases exponentially. [14] proposes a modified BB algorithm using generalized benders decomposition, [4] proposes a suboptimal solution based on the successive convex approximation (SCA) method, however, those are both centralized based methods which need to know the global prior knowledge of the network. Considering the mobility characteristics of network and limited resources of IIoT devices, it is hard to apply those methods in practice IIoT scenarios. Therefore, in this paper, we use the deep Q-learning framework to solve the problem distributedly and intelligently with low time complexity.

3 Proposed intelligent computation offloading solution: MDSPR

In this section, we present our solutions using multi-agent deep Q-learning with prioritized replay (MDPR) for the energy-efficent multi-agent computation offloading problem.

3.1 Markov decision process modeling

We first model the problem as a markov decision process (MDP). A MDP involves a agent that repeatedly observes the current state s_t during a time period. Agents interact with the environment through making actions a among all the available actions in that state, and the environment gives a feedback reward r_t to agents. Then, the agent will transfer to a new state s_{t+1} with a transition probability $P(s_{t+1}|s_t, a)$. A typical MDP tuple can be modeled as $<\mathcal{S}, \mathcal{A}, \mathcal{F}, \mathcal{R}, \mathcal{S}'>$.

3.1.1 State space

A state vector should reflect the agents' perception of the environment. According to our network model, we define $s_t = \{\mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3, \mathcal{S}_4\}$ as the state vector. $\mathcal{S}_1 = \{s_i(t), c_i(t), d_i(t)\}$ reflects the task properties in each time interval. $\mathcal{S}_2 = \{x_i(t), y_i(t)\}$, $\mathcal{S}_3 = \{X_1(t), Y_1(t), X_2(t), Y_2(t), \ldots, X_N(t), Y_N(t)\}$ denotes the current locations of the IIoT device and UAVs. $\mathcal{S}_4 = \{H_i(t), h_i^1(t), h_i^2(t), \ldots, h_i^N(t)\}$ gives the channel conditions between i-agent and different offloading objectives. The dimension of a state vector is $3N + 6$.

3.1.2 Action space

$\mathcal{A} = \{-1, 0, 1, 2, 3, \ldots, N\}$ represents the action space. Here $a = -1$ means to offload the task to cloud center, $a = 0$ means to execute the task locally, $a = \{1, 2, 3, \ldots, N\}$ denotes that the agent chooses to offload the task to UAVs.

3.1.3 Transition probability

$\mathcal{F}$ denotes the probability distribution $P(s_{t+1}|s_t, \{a\}_{a\in\mathcal{A}}) \in [0, 1]$ when state transition happens. Note that, in our network model, the agent cannot predict the state and reward of next state before making actions, so the MDP problem needs to be solved by model-free reinforcement learning method, in which the transition probabiliy distribution remains unknowm.

3.1.4 Reward function

The reward function is a immediate feedback for agents' actions from the environment. In our optimization problem, the objective is to minimize the total energy consumptions of agents, so the reward function should be inverse proportional to energy consumptions, which can be defined as

$$r(t) = \begin{cases} \dfrac{1}{I_i(t)E(t)}, & I_i(t)E(t) \le d_i(t) \\ \dfrac{1}{I_i(t)E(t)} - m, & I_i(t)E(t) > d_i(t) \end{cases} \tag{17}$$

where m is a punishment factor when the delay constraint is not satisfied.

3.2 Deep *Q*-learning structure

In this paper, we use deep Q-learning structure to solve the MDP. Deep Q-learing (DQN) is a promising reinforcement learning technique which combines deep learning with Q-learing aiming to solve complex problems in communication and networking [19]. Compared with Q-learning, DQN uses deep neural networks to replace the traditional Q-table, so that DQN can overcome the curse of dimensionality in Q-learning. The key idea of DQN is to use neural network as a function approximator. Given input state s_t, the output of neural network is $Q(s, a_i; \mathbf{w})$, where $\mathbf{w}$ is the weights of DQN.

Experience replay is an important technique to improve training efficiency in DQN. Since there is no Q-table in DQN, in a series of actions, the training samples may have strong correlations which will make learning process fall into local optimum. To solve this, DQN introduces a replay memory $\mathcal{D}$, once the agent obtain a new experience $< s_t, a_t, r_t, s_{t+1} >$, the experience is stored in $\mathcal{D}$. Each training step a mini-batch with size L is randomly sampled from D instead of a one step experience.

In the training stage, despite the current network $Q(s, a_i; \mathbf{w})$, DQN introduces another target network $Q(s, a_i; \mathbf{w}')$ to produce target value. The weights of $Q(s, a_i; \mathbf{w}')$ are updated every Z training steps. The target value is defined as

$$y_t = r_t + \gamma \max_{a'\in\mathcal{A}} Q(s', a'; \mathbf{w}'), \tag{18}$$

where γ denotes the reward decay in DQN. However, recent research has proved that the greedy iteration method may cause overestimation. Double DQN (DDQN) is proposed to eliminate overestimation which can be expressed as

$$y_t = r_t + \gamma Q(s', \operatorname*{argmax}_{a} Q(s, a_i; \mathbf{w}); \mathbf{w}'), \tag{19}$$

DDQN eliminates the problem of overestimation by decoupling the selection of target Q action and the calculation of target Q into two neural networks. Temporal difference error (TD-error) is defined as the difference between target value and current value which can be denoted as

$$\delta_t^j = |y_t - Q(s_t, a_1, a_2, \ldots, a_N)|, \forall t \in \mathcal{T}, \forall j \in \mathcal{N} \tag{20}$$

The loss function is defined as the least squared loss of TD-error denoted as

$$\mathbb{L}(\mathbf{w}) = \mathbb{E}[(y_{target} - Q(s, a; \mathbf{w}))^2]. \tag{21}$$

$\mathbb{L}(\mathbf{w})$ can be minimized through back propogation methods in deep learning, in this paper we use the classic gradient descent method

$$\mathbf{w} = \mathbf{w} - \alpha \bigtriangledown_{\mathbf{w}} \mathbb{L}(\mathbf{w}), \tag{22}$$

where α is the learning rate.

3.3 Stochastic prioritized replay

Traditional experienced replay method selects L experiences randomly from D, however, the importance of

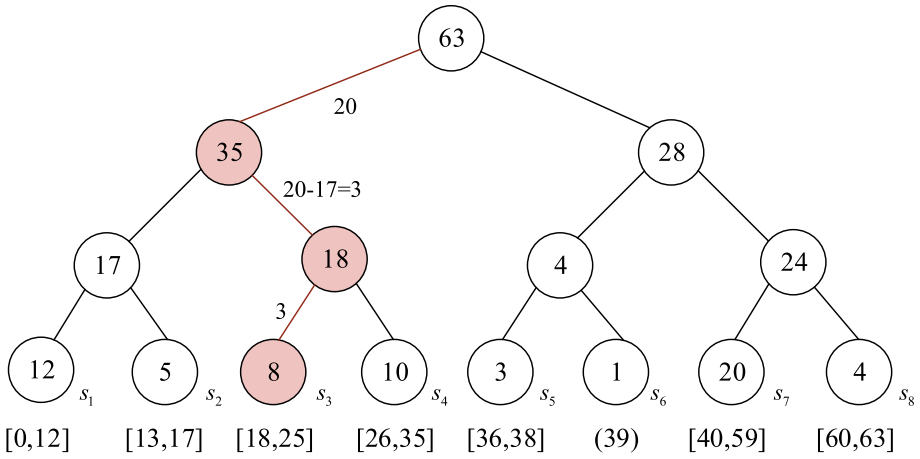

Fig. 2 Store experience priorities with binary tree structure

experiences in replay memory are different. Experiences with larger TD-errors may have better training effect [20]. Furthermore, simply choosing L experiences with the largest TD-errors (greedy prioritized) in each training step will cause high time complexity since the size of $\mathcal{D}$ is large. Therefore, according to [20], in this paper we use deep Q-learning with stochastic prioritised replay. We adopt the binary tree structure to store and sample experiences with priorities as shown in Fig. 2.

The top node stores the sum of all priorities in $\mathcal{D}$, in sampling stage, the sum is divided equally into L intervals. In sampling stage, each interval generates a random count in its range, and then starts to search downwards with the top node as the parent node. If the input value is less than the left child node, the left child node is regarded as the parent node to continue the downward search, otherwise, the input value is subtracted by the value of left child node, then, the right child node is denoted as the parent node, and the difference is used as an input to continue searching downwards until the last layer, and the experience corresponding to the parent node is the output sample. Figure 3 shows a typical sampling process with a input value 20. The chosen probability distribution of s_k can be denoted as

$$P(k) = \frac{p_k}{\sum_{d=1}^{O} p_d}. \tag{23}$$

where O denotes the capacity of $\mathcal{D}$. (12) shows that the higher the priority, the greater the probability of being selected. Moreover, the iterations of stochastic prioritized replay during each sampling process is $L * log_2(O)$, compared with greedy prioritized $L * O$.

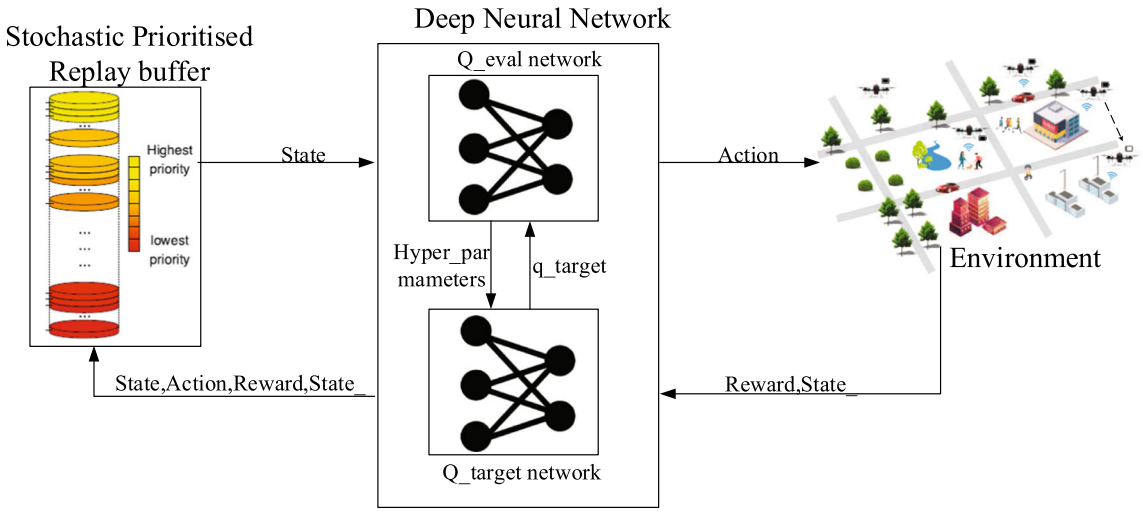

Fig. 3 The workflow of proposed MDSPR solution

Algorithm 1 Proposed solution: MDSPR

1: Initialize reward decay γ, learning rate α, greedy factor ϵ, batch size L, episode E, time peroid T, punishment factor m , step counter o, weights replaced step Z.
2: **for** device i in $\mathcal{M}$ **do**
3: Initialize $Q^i(s, a_i; \mathbf{w})$ with weights **w**, initialize target network $Q^i(s, a_i; \mathbf{w'})$ with weights **w'**, let **w'=w**.
4: **end for**
5: step counter $o = 0$
6: **for** episode e=$\{1, 2, 3, ..., E\}$ **do**
7: Initialize environment, get initial state s_0
8: **for** step t=$\{1, 2, 3, ..., T\}$ **do**
9: **for** device i in $\mathcal{M}$ **do**
10: Initialize task model $W_i(t) = (s_i(t), c_i(t), d_i(t))$
11: Select an action a_t^i based on ϵ greedy policy.
12: **if** $a_t^i \in \{1, 2, 3, ..N\}$ **then**
13: Compute the distance G between device i and UAV a_t^i.
14: **end if**
15: Compute the processing time q_t^i under a_t^i, and obtain reward r_t^i according to (14)
16: Store experience $< s_t, a_t^i, r_t^i, s_{t+1} >$ into $\mathcal{D}$ with random priority δ_t^i.
17: **end for**
18: Obtain reward $r_t := \left\{r_t^1, r_t^2, ..., r_t^M\right\}$ and new state s_{t+1}
19: $s_t = s_{t+1}$, $o = o + 1$
20: **if** $o > len(\mathcal{D})$ **then**
21: **for** device i in $\mathcal{M}$ **do**
22: Sample a mini-batch with L experiences from $\mathcal{D}$ according to stochastic prioritized replay method.
23: Update w according to (19), let **w'=w** in every Z steps.
24: Update δ_i^t according to (17)
25: **end for**
26: **end if**
27: **end for**
28: $\epsilon = \epsilon + 0.0001$
29: **end for**

3.4 Algorithm description: MDSPR

In this section, we propose our multi-agent deep Q-learning with stochastic prioritized replay alogrithm (MDSPR) for energy-efficient multi-user computation offloading problem. The algorithm workflow is shown in Fig. 3.

The pseudocode of our solution is shown in Algorithm 1. In the beginning, we initialize reward decay γ, learning rate α, greedy factor ϵ, batch size L, episode E, time peroid T, punishment factor m , step counter o and weights replaced step Z (Algorithm 1: Line 1). Then, for each IIoT device i, we bulid the current neural network $Q^i(s, a; \mathbf{w})$ and target neural network $Q^i(s, a; \mathbf{w}')$ (Algorithm 1: Line 2–4). Before training, we initialize step counter o (Algorithm 1: Line 5). In each episode, we initialize the environment and receive an initial state s_0, then the multi-agent computation offloading task that lasts for T steps starts (Algorithm 1: Line 7). In each training step, devices initialize the task model $W_i(t)$ and take actions based on ϵ-greedy policy (Algorithm 1: Line 10–11), if the action refers to offload the task to UAVs, then we compute the distance G between i to a_t^i-th UAV. After that, we compute the processing time q_t^i and reward r_t^i corresponding to a_t^i (Algorithm 1: Line 12–15). Then we store experiences into $\mathcal{D}$ with random priority δ_t^i (Algorithm 1: Line 16). Then the new state $s_t + 1$ is obtained from the environment, and we add the training step counter (Algorithm 1: Line 18–19).

Once the experience replay buffer is fulfilled, the learning stage starts. Each device i samples a mini-batch with L experiences from $\mathcal{D}$ according to stochastic prioritized replay method as input training samples. Then the weights of current network **w** are updated according to

(12), the weights of target network are updated by $\mathbf{w}$ every Z training steps, the priority values δ_i^t are updated according to (10) (Algorithm 1: Line 20–24). Finally, after each training step, ϵ is added with a small increment (Algorithm 1: Line 28).

In our work, neural networks with two hidden layers are applied as training network. We use ReLU function as activation function for each neurons. To avoid overfitting, the values of input state vector are mapped to the interval $[-1, 1]$.

4 Simulation results

In this section, we first give the detailed parameters settings in our simulation. Then, we present the benchmark algorithms used in computation offloading problem. Finally, we compare the performance of our proposed MDSPR with benchmark algorithms in terms of energy-efficiency and task successful ratio.

4.1 Parameters settings

We consider a smart factory scenario with a fixed area of 800 m × 800 m, devices need to deal with tasks generated by control center in every time interval t. Tasks can be either offloaded to UAVs or to the remote cloud center or execute locally. The number of IIoT devices M is set as 1000, the number of UAVs is set as 5 and the flying height H is 100 m. The communication range of each UAV R is set as 300 m. Devices are randomly distributed in the fixed area with local computation capablities C_l 500 MHz. The computation capablities of UAVs C_u and the cloud center C_c are set as 2 GHz and 100 GHz, respectively. The sizes of tasks are distributed within [100 Kb,100 Mb], the CPU circles required by tasks are generated randomly within $[5 \times 10^5, 5 \times 10^9]$, the tolerant delay of tasks are choosen in [0.01 s, 1 s]. We set the LoS channel power gain β_0 at the reference distance 1m as 1.42×10^{-4}. The bandwidth B is 15 MHz, w is 10MHz, the transmition power of MU to UAVs P_i is 0.01 W, the transmition power of MU to base stations P_i is 0.015 W and the noise power σ^2 is −90 dBm/Hz. The propogation time factor from MU to cloud server τ is 4.0×10^{-9}s/bit. The local energy consumption factor θ is set as 1×10^{-23}J/cycle. In this paper, we use a two-layer fully-connected neural network with each layer 100 neuron units as out training network. The learning rate α is 0.0001, the memory size of experiences is 10,000, the batch size is 50. The target replace steps Z is 500, ϵ is set as 0.9, the reward decay γ is set as 0.7. The main parameters are listed in Table I. All the simulation are run in Python 3.7 with Tensorflow 1.3 in Windows 10 operating system.

4.2 Performance analysis

First, we show the convergence performance of the MDSPR. Figure 4 shows the changes of loss over training steps, it is observed that the loss reduce rapidly at the beginning. This is because at the beginning of the training, agents take actions randomly and the approximation is not precise. After 100 steps the loss become stable, which indicates the convergence of MDSPR.

We then compare the performance of our MDSPR with three benchmark algorithms, namely random algorithm, DQN algorithm and DDQN algorithm. Figure 5 shows the average energy consumptions of a mobile user when dealing with a single task. Since the tasks are generated with random s_i, c_i, d_i, an inappropriate offloading decision may cause huge additional energy comsumptions. For example, if a task with high computation requirement and small size is executed in local, the cost will increase significantly compared with that the task offloaded to the remote cloud center. In the first n training steps, the agent makes offloading decisions according to ϵ-greedy policy, after the learning stage, the average energy consumptions become stable due to the convergence of neural networks.

Figure 6 shows the successful task execution ratio versus training steps. In this paper, in the following two cases, the execution of a task is regarded as a failure. The first case is the execution time exceeds the tolerant time d_i, which means the task cannot be finished in the required QoS. The second case is in UAVs' offloading, if a task is choosen to offload to a UAV, and the distance between the mobile user and the target UAV is larger than the communication range of UAVs. In Fig. 6, the successful task execution ratio of MDSPR outperforms the benchmark algorithms, which indicates MDSPR has a higher probability to make energy-efficient decisions.

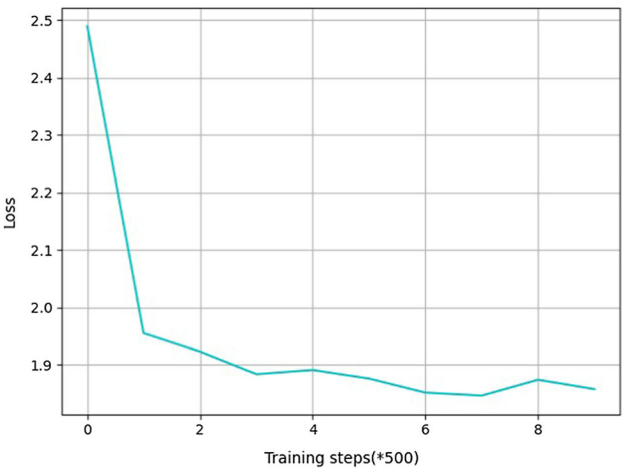

Fig. 4 The convergence performance of MDSPR

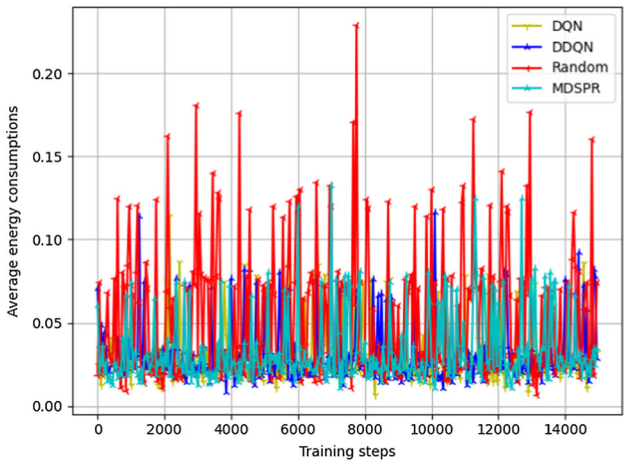

Fig. 5 Average energy consumption of IIoT devices with different algorithms when processing a single task

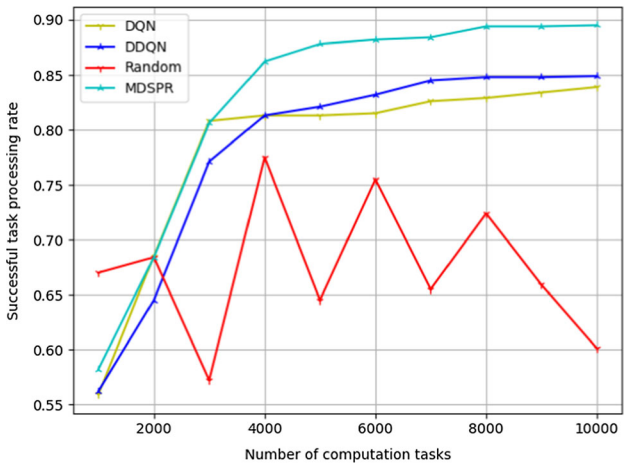

Fig. 6 The task successful rate versus the number of computation tasks under different offloading strategies

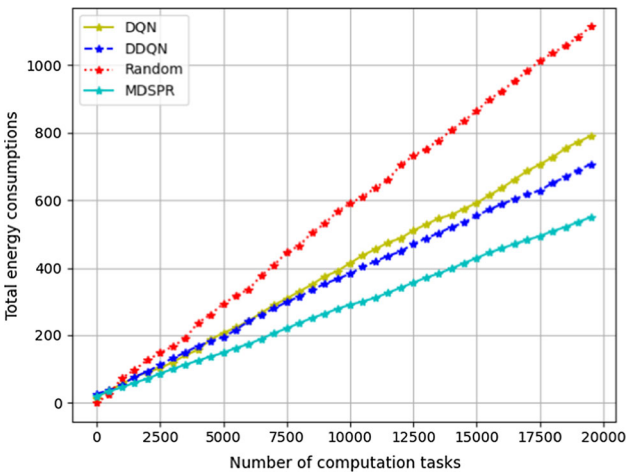

Fig. 7 Total energy consumptions upon a single device handling different numbers of tasks

Figure 7 shows the total energy consumptions under different number of tasks after the learning stage,. From Fig. 7, it can be seen that the proposed MDSPR outperforms the three benchmark algorithms. The reason is that in each training step, MDSPR chooses more valuable experiences to learn, thus maximizing the overall long-time reward. Furthermore, it is observed that with the increase of tasks, the energy consumptions increase approximately linearly. This is because agents make offloading decisions upon each task arrival according to the maximum Q-value. After the convergence of the neural network, the action with highest Q-value has a high probability to obtain minimize energy consumptions.

Figure 8 shows the total energy consumptions upon different number of devices, each device processes 500 tasks. The states of devices are initialized randomly at the beginning, it can be seen that the proposed MDSPR outperforms the benchmark algorithms. Furthermore, Fig. 8 indicates that the time complexity of the proposed MDSPR algorithm with respect to moblie users is n, which means as the number of mobile users increase, the total energy consumptions increase linearly. This shows that the proposed MDSPR algorithm can enable different users to make decisions distributedly, with the possibility of being applied in large-scale scenarios.

5 Discussions

5.1 Analysis of MDSPR algorithm complexity

MDSPR can be divided into two stages, namely learning stage and application stage. In the learning phase, when the memory pool is full, the algorithm uses random priority playback to select a total of L experience samples for

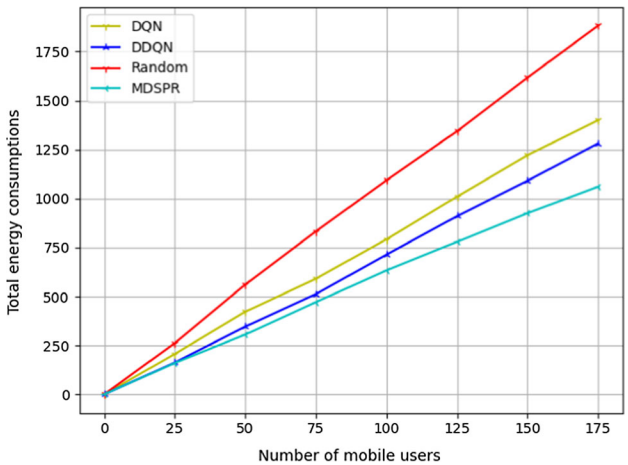

Fig. 8 Total energy consumptions upon different devices under a total of 500 tasks

training for each user in each training period and each time duration period. Due to the principle of binary tree search, therefore, the total number of iterations required to select a sample is $L \times \log_2(O)$, Since there is a certain probability in ϵ-greedy policy to select the offloading object corresponding to the maximum value of all output values as the offloading decision, the optimal complexity in the existing sorting algorithm is $O(n)$, so the total number of iterations required to select an action is $\epsilon \times (N+2)$, so the learning phase The time complexity is: $E \times T \times M \times L \times \log_2(O) \times \varepsilon \times (N+2)$, in the application phase, since the network has converged, there is no need to select empirical samples for training, and $\epsilon = 1$ only the action output corresponding to the maximum value from all output values needs to be taken. The time complexity of the application phase is $E \times T \times M \times (N+2)$. It can be seen that the algorithm complexity of the learning phase is much higher than that of the application phase. Therefore, the control center with strong computing caapcity can accept the parameters of each agent to complete the whole learning phase, and then copy the parameters of the neural network and transfer them to the local IIoT devices. Then, MDSPR algorithm can be applied in the device distributedly.

As for comparison, we further discuss the complexity of the traversal algorithm, random algorithm and DQN, DDQN algorithm. In order to find the optimum solution, the traversal algorithm needs to calculate the energy consumptions and processing time of one task offloading to each offloading destination, which is more than $(N+2)$ the total number of iterations compared with the MDSPR algorithm. However, since the processing time of the task is unknown until the task is completed, since a task can only be offloaded to one offloaded destination or processed locally. Therefore, besides its high complexity, the traversal algorithm is unrealistic in practical applications. The random algorithm randomly selects an offloading strategy for each task, so the total complexity is $E \times T \times M$. According to the simulation results, the random algorithm has the lowest complexity but the worst performance. Both DQN and DDQN randomly select L training samples from the experience memory pool. Therefore, the complexity of the two in the learning phase is $\times T \times M \times \varepsilon \times (N+2)$ less than MDSPR, and the complexity of the application phase is the same as that of MDSPR, both are $E \times T \times M \times (N+2)$. Therefore, the MDSPR algorithm can be understood to some extent as the use of computing resources in exchange for performance.

5.2 Analysis of MDSPR algorithm effectiveness

Due to the dynamic nature of the task model in this article, at different time intervals, the size of each task, the number of required CPU circles and the maximum tolerent delay are different, resulting in different tasks being adapted to different offloading objects. Obviously, in an ideal state, the traversal algorithm can obtain the optimal calculation decision, that is, for each task, calculate its energy consumption in all possible offload objects and then take the minimum. However, the traversal algorithm has been discussed in 5.1. At large, the complexity is too high to be practically applied. The random algorithm has the lowest complexity, but it is prone to unloading decisions with poor performance, resulting in high energy consumption or even failure in task processing. The DQN and DDQN algorithms select the unloading object corresponding to the maximum value in the current network output value as the unloading decision. Since the calculation of the target value includes the return value, the network output value can reflect the energy consumption of the unloading decision, and the performance is better than random algorithm. On the basis of DQN and DDQN, the MDSPR algorithm selects samples with higher priority for learning, avoiding the interference of samples with low returns on the network output value, and further optimizing the performance.

However, it can be seen from Figs. 7 and 8 that the MDSPR algorithm cannot guarantee to make the most correct choice for each task even after convergence. By observing the estimated value of the current network output, it can be found that some of the output values have very small differences. At this time, taking the action with the largest output value as the offloading decision does not necessarily lead to the optimal offloading performance. However, under the guidance of rewards, although the action at this time is not optimal, it is still a suboptimal solution, and it will not make bad decisions frequently like random algorithms. Therefore, the performance of the MDSPR algorithm is between the traversal algorithm and the random algorithm. At the same time, the random priority selection strategy improves the time complexity of the MDSPR algorithm compared with DQN and DDQN, but the improvement is not much. Therefore, the strategy of central training and subsequent parameter replication and decentralization can be used to solve this problem.

5.3 Discussions on the application of deep reinforcement learning in wireless communication

Deep reinforcement learning was proposed by David Silver and other scholars in 2015. On the basis of reinforcement learning, deep neural networks are combined to solve the problem of traditional state space limitation. Typical applications include Atari games and the famous AlphaGo. At present, it is also widely used in the field of robot motion control. At present, more and more scholars are

exploring the application of deep reinforcement learning in wireless communication , including power control, access selection, spectrum utilization, routing decision, wireless caching, computing offloading, information collection and other proposals. Although the scenarios are different, the essence is that agents (users, devices, control centers and other entities) make decisions that are most beneficial to the overall revenues in a specific environment. With the development of deep reinforcement learning, the output of decision-making behavior can be discrete or continuous. Taking computation offloading issue as an example, the main research of this paper focuses on the binary computing unloading direction, so the DQN architecture in deep reinforcement learning is used to output the specific unloading object number, and its value is discrete. If further consider partial offloading, which is, the agent can divide a task into arbitrary parts, each part is composed of a certain number of bits and CPU circles, the maximum tolerant delay of the entire task remains unchanged, and the decision-making behavior becomes the device offloading to different offloading The number of bits of the object is a continuous value. At present, deep reinforcement learning has a continuous state and output action architecture represented by DDPG to support richer and more complex applications.

However, deep reinforcement learning still have great challenges when applied into wireless communication field, which are mainly manifested in two aspects. The first is whether the establishment of the environment fits the actual situation. In deep reinforcement learning, the role of the environment is to feedback a return value based on the input action and make the current state transmit to the next state. Therefore, the accuracy of the establishment of the environment reflects whether the reward value can represent the pros and cons of the current action, thereby affecting the learning result. The second is the convergence problem of neural networks. Taking traditional deep learning supervised learning as an example, the input samples are all with specific labels, so when backpropagating, only the difference between the output of the neural network and the label value is required for each neural network The partial derivative of the parameters can make the network converge. Even high-dimensional input samples such as images have limited characteristics. The dimensionality can be reduced by methods such as pooling and PCA. However, in deep reinforcement learning, due to the high dynamics of the environment, the sample state during each training is different, and the actions taken in each state are different. The target value of the target network output is simply the same as the current network output. The difference of the target value as the basis of backpropagation can easily lead to the overfitting of the network. Therefore, it is necessary to sparse the reward, normalize the input state, and adjust the environmental parameters and neural network related parameters for different applications. At present, many scholars are carrying out related researches on the convergence of deep reinforcement learning, and have achieved certain results by improving the learning structure and setting up reasonable returns.

6 Conclusion

In this paper, we study the multi-UAV-Enabled computation offloading problem in IIoT scenarios. We formulate the problem as an MINLP problem which is difficult to find global optimum through existing methods. In order to solve the problem efficiently, we propose our MDSPR method, which makes improvement on the basis of DQN. We first model the whole process as an MDP by defining state vectors, actions and reward functions. Then, we train the neural network with prioritized samples in experience replay. Simulation results show the MDSPR method out performs the random method, DQN method and DDQN method in terms of energy-efficiency and task successful rate. Further discussions show the effectiveness of deep reincforcement learning applying in wireless communication networks. As for future works, we tend to study the partial computation offloading problem and resource allocation problem by using more complex deep reinforcement learning methods.

References

1. Ojo, M. O., Giordano, S., Adami, D., & Pagano, M. (2019). Throughput maximizing and fair scheduling algorithms in industrial internet of things networks. *IEEE Transactions on Industrial Informatics, 15*(6), 3400–3410.
2. Willig, A. (2008). Recent and emerging topics in wireless industrial communications: a selection. *IEEE Transactions on Industrial Informatics, 4*(2), 102–124.
3. Mao, Y., You, C., Zhang, J., Huang, K., & Letaief, K. B. (2017). A survey on mobile edge computing: the communication perspective. *IEEE Communications Surveys & Tutorials, 19*(4), 2322–2358.
4. Jeong, S., Simeone, O., & Kang, J. (2018). Mobile edge computing via a UAV-mounted cloudlet: optimization of bit allocation and path planning. *IEEE Transactions on Vehicular Technology, 67*(3), 2049–2063.
5. Zhang, J., et al. (2019). Stochastic computation offloading and trajectory scheduling for UAV-assisted mobile edge computing. *IEEE Internet of Things Journal, 6*(2), 3688–3699.
6. Zhang, J., et al. (2020). Computation-efficient offloading and trajectory scheduling for multi-UAV assisted mobile edge computing. *IEEE Transactions on Vehicular Technology, 69*(2), 2114–2125.
7. Zhou, F., Wu, Y., Hu, R. Q., & Qian, Y. (2018). Computation rate maximization in UAV-enabled wireless-powered mobile-edge

computing systems. *IEEE Journal on Selected Areas in Communications, 36*(9), 1927–1941.
8. Mnih, V., et al. (2015). Human-level control through deep reinforcement learning. *Nature, 518*(7540), 529–533.
9. Wang, Y., Wang, K., Huang, H., Miyazaki, T., & Guo, S. (2019). Traffic and computation co-offloading with reinforcement learning in fog computing for industrial applications. *IEEE Transactions on Industrial Informatics, 15*(2), 976–986.
10. Liu, Y., Yu, H., Xie, S., & Zhang, Y. (2019). Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks. *IEEE Transactions on Vehicular Technology, 68*(11), 11158–11168.
11. Chen, X., Zhang, H., Wu, C., Mao, S., Ji, Y., & Bennis, M. (2019). Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning. *IEEE Internet of Things Journal, 6*(3), 4005–4018.
12. Lu, H., He, X., Du, M., Ruan, X., Sun, Y., Wang, K. Edge QoE: Computation offloading with deep reinforcement learning for internet of things, *IEEE Internet of Things Journal*, https://doi.org/10.1109/JIOT.2020.2981557
13. Hemmecke, R., Köppe, M., Lee, J., & Weismantel, R. (2012). Nonlinear integer programming. *Comput. Math. Appl., 1*(2), 215–222.
14. Zhang, J., et al. (2019). Joint resource allocation for latency-sensitive services over mobile edge computing networks with caching. *IEEE Internet of Things Journal, 6*(3), 4283–4294.
15. Al-Hourani, Akram Kandeepan. (2014). Optimal LAP altitude for maximum coverage. *IEEE Wireless Communications Letters, 3*(6), 569–572.
16. Marzetta, T. L. (2010). Noncooperative cellular wireless with unlimited numbers of base station antennas. *IEEE Trans. Wireless Commun., 9*(11), 3590–3600.
17. Guo, H., & Liu, J. (2018). Collaborative computation offloading for multiaccess edge computing over fiber-wireless networks. *IEEE Transactions on Vehicular Technology, 67*(5), 4514–4526.
18. Hu, X., Wong, K., Yang, K., & Zheng, Z. (2019). UAV-assisted relaying and edge computing: scheduling and trajectory optimization. *IEEE Transactions on Wireless Communications, 18*(10), 4738–4752.
19. Luong, N. C., et al. (2019). Applications of deep reinforcement learning in communications and networking: a survey. *IEEE Communications Surveys & Tutorials, 21*(4), 3133–3174.
20. Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2016). Prioritized Experience Replay, *ICLR*
21. Zhang, X., Zhang, J., Xiong, J., Zhou, L., & Wei, J. (2020). Energy-efficient multi-UAV-enabled multiaccess edge computing incorporating NOMA. *IEEE Internet of Things Journal, 7*(6), 5613–5627.
22. Peng, H., & Shen, X. (2021). Multi-agent reinforcement learning based resource management in MEC- and UAV-assisted vehicular networks. *IEEE Journal on Selected Areas in Communications, 39*(1), 131–141.

Shuo Shi is an Associate Professor with the Communication Research Center, Harbin Institute of Technology (HIT), Harbin, China. He received the B.E., M.S. and Ph.D. degrees in information and communication engineering from HIT, in 2002, 2004 and 2008, respectively. He studied as an exchange student at Network Research Lab of Sungkyunkwan University, Korea, from 2004 to 2005.His current research interests include mobile wireless Ad-Hoc network, signal detection and network architecture research of space vehicle.

Meng Wang received the bachelor's degree from Harbin Institute of Technology (HIT), Harbin, China in 2018. He is currently pursuing the Ph.D. degree with School of Electronic and Information Engineering in Harbin Institute of Technology. His current research interests include mobile-edge computing, UAV communication networks and AI in wireless communication.

Shushi Gu received the M.S and Ph.D. degrees in information and communication engineering from Harbin Institute of Technology in 2012 and 2016, respectively. Currently, he is an Assistant Professor with school of electronic and information engineering in Harbin Institute of Technology (Shenzhen), Shenzhen, China. From 2016 to 2019, he was a Postdoctoral Research Fellow with HITSZ. From 2018 to 2019, he was a Postdoctoral Research Fellow with James Cook University, Cairns, Australia. He received the Best Paper Awards of IEEE WCSP 2015 and EAI WiSATS 2019. He received the Outstanding Postdoctoral Award of HITSZ in 2018. His current research interests include satellite IoT, coding theory, edge caching and distributed storage.

Zhong Zheng received his B.S., M.S. and Ph.D. degrees in Science of Automation, Electronic Information Science and Technology, and Control Science and Engineering from Harbin Institute of Technology, Harbin, China, in 2006, 2008 and 2017, respectively. In 2008, he joined Harbin Institute of Technology as a Secretary for Headmasters. His current research interests include Markovian jump systems, fault detection, PWM converter control.