Service Management and Energy Scheduling Toward Low-Carbon Edge Computing

Lin Gu, *Member, IEEE*, Weiying Zhang, Zhongkui Wang, Deze Zeng, *Member, IEEE*, and Hai Jin, *Fellow, IEEE*

Abstract—Edge computing has become an alternative low-latency provision of cloud computing thanks to its close-proximity to the users, and the geo-distribution nature of edge servers enables the utilization green energy from the environment on-site. To pursue the goal of low-carbon edge computing, it is desirable to minimize the operational expenditure by scheduling the computing resource and green energy according to the spatially and temporally varying user demands. In this article, inspired by the successful application of deep reinforcement learning (DRL) in diverse domains, we propose a DRL-based edge computing management strategy which continuously explores the states and adaptively makes decisions on service management and energy scheduling, towards long-term cost minimization. Different from model-based solutions, our proposal is a model-free method, without any assumption on statistical knowledge as a priori, and therefore is practical in implementation. To speedup the agent training procedure, we further design a prioritized replay memory by utilizing the model-based solution as a guideline to set the transition priority. Extensive experiment results based on real-world traces validate that our proposed DRL-based strategy can make considerably progress compared to the one-shot greedy strategy, and it can learn the system dynamically to manage the edge computing services at runtime.

Index Terms—Low-carbon edge computing, deep reinforcement learning, service management

1 Introduction

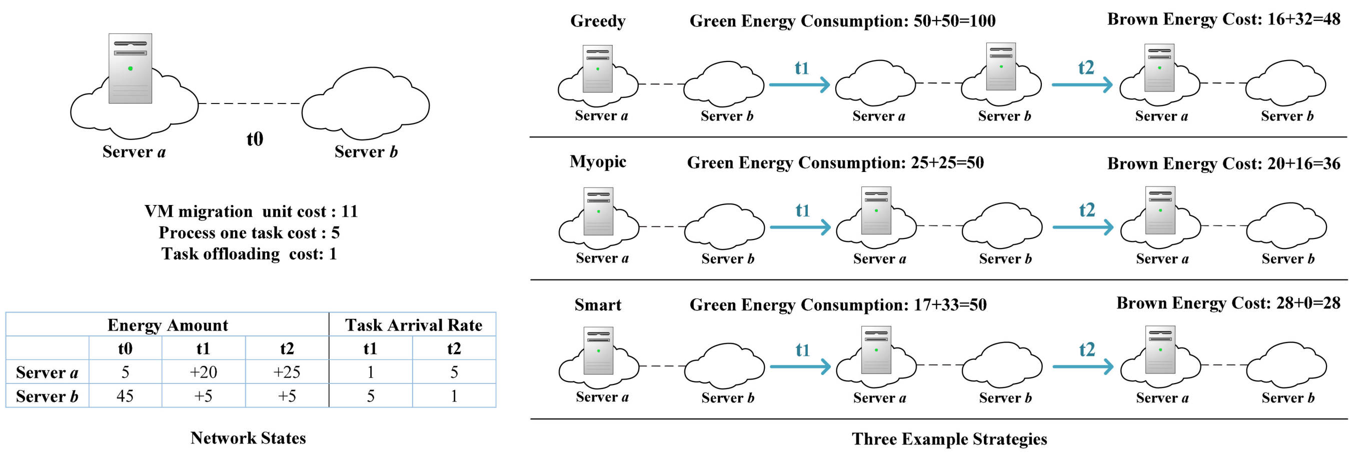
With the exponential growth of computing demands, it is increasingly difficult for traditional cloud computing to meet the requirements of various latency-sensitive applications (e.g., mobile applications, Internet-of-Things, connected and autonomous vehicles). Edge computing has been becoming an attractive low-latency computing paradigm by pushing services to the edge servers within the user proximity, and is promising in migrating the heavy burden from both the cloud computing platform and the backbone networks. However, as computing demand grows, so does the electricity consumption and the emission of greenhouse gas. Low carbon cloud computing has been proposed to limit the carbon emissions of cloud by many studies [1], [2]. For example, Canada's Greenstar Network [1] connects cloud computing resources directly to wind turbines or solar power generators, using green energy to reduce carbon emissions. [3] and [4] study the grid power market records and propose a computing model that directly connects cloud computing resources and green energy generator to reuse the wasted energy during over-supply and transmission congestion.

Compared with the cloud, edge computing shows great advantages in utilizing green and sustainable computing because its geo-distributed servers can naturally harvest various on-site green energy (e.g., wind energy, hydroelectric energy, and solar energy) as the first power supply [5], [6], [7], [8], to realize low-carbon edge computing. However, edge computing is more complex than cloud computing, due to its higher dynamics in a wide range of aspects, e.g., user mobility, user demands, resource availability. Specifically, the distribution of user demands is usually related to time and user mobility, and the service s should also be migrated accordingly. Green energy, featured by unpredictability and geographical diversity, further exaggerates such dynamics. To ensure energy supply stably, it is not enough to exclusively utilize intermittent green energy, so electricity from power grid is needed. But grid company usually charges electricity in the form of step tariff, i.e., time-varying price, which cannot be ignored.

From the perspective of edge computing operators, it is always desirable to reduce the *operational expenditure* (OPEX) including energy cost as one of the most dominant parts, so as to maximize revenue. Therefore, how to jointly manage the services and schedule the energy in edge computing toward the goal of OPEX minimization has been becoming a hot research topic recently. For example, Yang et al. [9] propose an offline service management strategy with the assumption that green energy cannot be stored, to minimize short-term brown energy costs. Mao et al. [10] invent a

- *Lin Gu, Weiying Zhang, Zhongkui Wang, and Hai Jin are with the National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, Hubei 430074, China. E-mail: {lingu, zhang_wy1, zkwang, hjin}@hust.edu.cn.*
- *Deze Zeng is with the Hubei Key Laboratory of Intelligent Geo-Information Processing, School of Computer Science, China University of Geosciences, Wuhan, Hubei 430074, China. E-mail: deze@cug.edu.cn.*

Manuscript received 26 March 2022; revised 17 June 2022; accepted 7 August 2022. Date of publication 29 September 2022; date of current version 7 March 2023.
This work was supported in part by the NSF of China under Grants 61972171, 62172375, 62232011, and in part by the Open Research Projects of Zhejiang Lab under Grant 2021KE0AB02.
(Corresponding author: Deze Zeng.)
Recommended for acceptance by H. Yao.
Digital Object Identifier no. 10.1109/TSUSC.2022.3210564


Server a
t0
Server b
VM migration unit cost : 11
Process one task cost : 5
Task offloading cost: 1
Network States
Greedy
Green Energy Consumption: 50+50=100
Brown Energy Cost: 16+32=48
Myopic
Green Energy Consumption: 25+25=50
Brown Energy Cost: 20+16=36
Smart
Green Energy Consumption: 17+33=50
Brown Energy Cost: 28+0=28
t1
t2
Server a
Server b
Three Example Strategies

	Energy Amount			Task Arrival Rate	
	t0	t1	t2	t1	t2
Server *a*	5	+20	+25	1	5
Server *b*	45	+5	+5	5	1

Fig. 1. An example on service management and energy scheduling.

Lyapunov optimization based algorithm to reduce the service cost of edge computing by aggressively maximizing the utility of available green energy. Chen et al. [11] also propose an online peer offloading strategy that maximizes the performance efficiency while guaranteeing the long-term energy consumption constraints on edge servers.

Although there have been many researches in this area, we notice that most proposed methods are based on optimization models with certain assumptions or prerequisites to manage service. Even there are some online algorithms, they usually oversimplify the complex system or model it inaccurately. When these model-based optimization methods are applied in practice scenarios, the achieved performance is usually not satisfactory. Therefore, a model-free solution is desired to intelligently manage service according to runtime system dynamics. Fortunately, the successful application of AlphaGo series on game control has been raising fervent concern of academia and industry to *reinforcement learning* (RL) again [12], [13]. Especially with the combination of deep learning, *deep reinforcement learning* (DRL) and its derivative algorithms such as *prioritized experience replay* (PER-DQN) [14] have already been extensively used in many control domains, e.g., robotics, autonomous driving, and traffic light control, with no exception to the ICT system management. For example, Huang et al. [15] propose a *deep reinforcement learning-based online offloading* (DROO) framework for offloading decisions and wireless resource allocation with the goal of maximizing the weighted sum computation rate. Therefore, applying RL to design a model-free service management strategy for edge computing is a feasible way.

However, these solvers are shown to be inefficient when they come to complex environments. Take DROO as an example, the solver adopts a deep neural network to learn the integer variables, it will be time consuming with the increment of action space. In this paper, we design a service management strategy from the perspective of edge computing operators towards the goal of long-term cost efficiency. To achieve this, there are a series of different spatial and temporal dynamics that need to be addressed, as declared before, including the user demands, green energy generation, and brown energy pricing. Usually, the service runs in the form of container, which can be freely migrated among different edge servers. In pursuit of cost efficiency, the operator shall make online decisions on both the service management (i.e., which edge server to migrate the service) and the energy scheduling (i.e., how much green energy shall be utilized) in response to the real-time system environment and potential future dynamics. This naturally fits in the application scope of RL. Therefore, we leverage the representative DRL algorithm *deep Q-network* (DQN), to design an efficient energy-aware service management strategy. We mainly make the following contributions in this paper:

- With the goal of minimizing the long-term OPEX, we formulate the online service management and energy scheduling problem into a *mixed integer linear programming* (MILP) form.
- We consider the effect of current service migration and energy scheduling actions for future energy consumption and leverage the DQN technology and design a DQN based solution to reduce migration cost and improve green energy efficiency.
- To further accelerate the DQN agent training, we propose a *prioritized DRL* (pDRL) algorithm by redesigning a prioritized episodic transition sampling method according to the *temporal-difference* (TD) error and utilizing the solution of the MILP problem as a guideline to improve the efficiency of the the actions.
- Extensive experiments are conducted to show that our pDRL accelerates the convergence speed by 37.67% and reduces the long-term OPEX by 32.70%, compared with state-of-the-art studies.

The reminder of this paper is organized as follows. First, section 2 introduces the investigated problem through a toy example and presents the formal problem formulation. Our customized DRL algorithm is proposed in Section 3. Then, Section 4 illustrates the experimental results and 5 discusses some related studies. Finally, the conclusion of this paper is provided in Section 6.

2 Motivation and Problem Statement

2.1 Motivation Example

To understand this problem better, let us consider a toy example of service management and energy scheduling

TABLE 1
Major Notations

Constants	
$\mathbb{N}$	The set of edge node n
$\mathbb{F}$	The set of different type of service f
$\mathbb{T}$	The set of time slot t
H_{mn}	The hop from the edge node m to another edge node n
P_f	The energy consumption of request processing for service f in a unit
S_f	The energy consumption of request transmission for service f in a unit
V_f	The energy consumption of service migration for service f in a unit
$\lambda_f^n(t)$	The request arrival rate of service f in edge node n at time slot t
$G_n(t)$	The green energy reserved in edge node n at time slot t
Variables	
$x_f^n(t)$	Binary variables to indicate whether edge node n host service f
$\mu_n(t)$	The total amount of green energy used for all requests in edge node n at time slot t
$e_n(t)$	The total amount of energy needed for all requests in edge node n at time slot t
$r_n(t)$	The total amount of reserved green energy in node n at the end of t

problem in two edge servers over 2 time slots, as shown in Fig 1. In this example, we consider the total cost consisting of three aspects: 1) processing cost: Only the edge servers with the corresponding service can process the requests at an energy cost of 5. We consider a simple scenario that there is only one node to run the service while others are hibernating to save energy. 2) service migration cost: Migrating service between edge servers costs 11 units of energy in the node hosting service; and 3) communication cost: Service demands need to be routed to the edge server hosting service with a cost of 1 per request per hop. Initially, service is located at edge server a, and each server reserves a certain amount of green energy, i.e., 25 units in edge server a, and 50 units in edge server b. And the number of hops between server a and b is 3. The newly generated green energy amounts are $\{20, 25\}$ and $\{5, 5\}$ and the request arrival rates are $\{1, 5\}$ and $\{5, 1\}$ in server a and b, at time slots t_1 and t_2, respectively. Assume the brown energy costs is 1 at time slot t_1 and 2 at time slot t_2, while the green energy is free.

One of the straightforward solutions is maximizing the utility of green energy at each time slot in a **Greedy** way. That is, we move the service to server b at time slot t_1 with 50 units green energy cost and then migrate back to server a at time slot t_2 with 50 units green energy cost, i.e., choosing the edge servers with maximum green energy. We can see from strategy **Greedy** that $30 + 33 + 3 = 66$ units of energy are needed by server b at time slot t_1 while 50 units can be covered by local green energy. So the energy cost is $(66 - 50) \cdot 1 = 16$. Similarly, the energy cost of service in server a at time slot t_2 is $16 \cdot 2 = 32$. In this case, the **Greedy** strategy produces a total energy cost of 48 as shown in Fig. 1. From the **Greedy** strategy, we can see that greedily using the green energy might need to migrate the service very often, hence resulting in a higher energy cost. So, now we consider leaving service in edge server a, using its available green energy as shown in strategy **Myopic**. In this case, at time slot t_1 and t_2, the brown energy consumptions are $(30 + 15) - 25 = 20$ and $(30 + 3) - 25 = 8$, receptively, leading to a total cost of $20 \cdot 1 + 8 \cdot 2 = 36$. It is noticeable that the brown energy price at time slot t_2 is 2, which is relatively higher than time slot t_1. So, if we are smart enough, we should save the green energy at time slot t_1 to t_2 for future use and reduce the total energy efficiently. The **Smart** strategy in Fig. 1 shows that total energy consumption is not changed, yet we use only part of the green energy in server a at t_1 as 17 and leave the rest to t_2, finally, the total brown energy cost reduced to $28 \cdot 1 + 0 \cdot 2 = 28$.

Comparing with the results of **Greedy** and **Myopic**, we can find that greedily selecting edge server with the highest on-site green energy supply at each time slot will not always find a satisfactory solution. Similarly, from **Myopic** and **Smart**, it can be obtained that aggressively using all the green energy in edge server might miss the opportunity to minimize the long-term energy cost.

2.2 Problem Statement and Formulation

Based on the toy example, now let us consider a general case of energy scheduling problem in a set of edge nodes $\mathbb{N}$ over discrete time slots $\mathbb{T} = \{1, 2, 3, ..., T\}$. Table 1 shows the major notations used in this paper.

2.2.1 Service Migration and Energy Consumption

To achieve low carbon edge computing, we need to select the proper edge node to host the service and decide the utilized green energy amount at each time slot. This long-term energy cost minimization over time period $\mathbb{T}$ can be considered as a *Markov decision process* (MDP). Note that certain demands can only be performed on the edge node hosting the corresponding service. So we are interested in reducing the total energy cost consisting of three parts: 1) request transmission, 2) request processing, and 3) service migration.

Let binary variables $x_f^n(t)$ indicate whether the service f is placed in edge node n at time slot t, as

$$x_f^n(t) = \begin{cases} 1, & \text{whether service f is placed in n at time slot t,} \\ 0, & \text{otherwise.} \end{cases}$$

Note that, the amount of arrival requests for different services is variable at different time slot, which contributes to the diversity of requests arrival rate. Besides, the newly generated green energy also varies on time and location. At the beginning of each t, we need to decide how to place the services and schedule the energy. For each type of service f, we consider one service instance for its requests, i.e.,

$$\sum_{n \in \mathbb{N}} x_f^n(t) = 1. \tag{1}$$

Once an edge node n is selected, all the offloaded requests then will be transformed and processed in its corresponding service on node n. That is, if $x_f^n(t) = 1$, edge node n is selected to process the requests of service f with rate $\sum_{m \in \mathbb{N}} \lambda_f^m$ and the energy consumption of request processing P_f , thus the processing energy consumption of node n is

$$e_n^p(t) = \sum_{f \in \mathbb{F}} \sum_{m \in \mathbb{N}} x_f^n(t) \cdot \lambda_f^m(t) \cdot P_f, \tag{2}$$

also, the transmission energy cannot be ignored. Since requests $\lambda_f^n(t)$ need to be transformed from edge node m to n with energy consumption of request transmission S_f, therefore, the transmission energy consumption $e_n^s(t)$ as

$$e_n^s(t) = \sum_{f \in \mathbb{F}} \sum_{m \in \mathbb{N}} x_f^n(t) \cdot \lambda_f^m(t) \cdot H_{mn} \cdot S_f. \tag{3}$$

As for the service migration cost $e_n^v(t)$, it is related to the location of service at previous time slot $x_f^m(t-1)$ and the "distances" of migration, i.e., network hops H_{mn}. When a service migration decision is made from edge node m to n, $m, n \in \mathbb{N}$, it will consume energy V_f for service migration, thus, the migration energy consumption $e_n^v(t)$ occurs as

$$e_n^v(t) = \sum_{f \in \mathbb{F}} \sum_{m \in \mathbb{N}} x_f^n(t) \cdot x_f^m(t-1) \cdot H_{mn} \cdot V_f. \tag{4}$$

Note that, if no service migration happens, the migration cost $e_f^v(t) = 0$ since the migration distance H_{mn} is 0.

Notice that equation (4) is nonlinear due to the products of two binary variables, i.e., $x_f^n(t) \cdot x_f^m(t-1)$. To linearise this equation to linear, a new binary variable $y_f^o(t), o \in \mathbb{N}$ is defined as

$$y_f^o(t) = x_f^n(t) \cdot x_f^m(t-1), \forall (m,n) \in \mathbb{N}, \tag{5}$$

then the products can be equivalently replaced by the new variables with the following two linear constraints as

$$0 \le y_f^o(t) \le x_f^n(t), \forall (m,n) \in \mathbb{N}, \tag{6}$$

and since $x_f^n(t)$ is a two binary variable, therefore

$$x_f^n(t) + x_f^m(t-1) - 1 \le y_f^o(t) \le x_f^m(t-1), \forall (m,n) \in \mathbb{N}. \tag{7}$$

In this case, (4) can be rewritten into a linear equation as

$$e_n^v(t) = \sum_{f \in \mathbb{F}} \sum_{m \in \mathbb{N}} y_f^o(t) \cdot H_{mn} \cdot V_f, \tag{8}$$

with constraints (6) and (7).

2.3 Green Energy Scheduling

There is no doubt that the generation of green energy varies from node to node and time to time. For example, solar powered edge servers can harvest sufficient green energy in sunny day while others may not. If there is not enough green energy to process all requests on the selected node n, brown energy will be utilized at a higher price. Since aggressively using up the green energy in selected edge node n with service may cause a higher brown energy cost during future peak hours, we need to determine the number of used green energy at the start of each time slot t as $\mu_n(t)$. Note that we can only schedule the green energy of selected edge node hosting the service, i.e.,

$$\frac{\sum_{f \in \mathbb{F}} x_f^n(t)}{A} \le \mu_n(t) \le \sum_{f \in \mathbb{F}} x_f^n(t) \cdot A, \forall n \in \mathbb{N}, \tag{9}$$

where A is an arbitrarily large number.

Of course, the green energy amount also cannot exceed the available amount or the total energy required. Let $e^n(t)$ be the total energy requirement as

$$e_n(t) = e_n^p(t) + e_n^s(t) + e_n^v(t), \forall n \in \mathbb{N}, \tag{10}$$

Then we have

$$0 \le \mu_n(t) \le G_n(t), \forall n \in \mathbb{N}, \tag{11}$$

and

$$0 \le \mu_n(t) \le e_n(t), \forall n \in \mathbb{N}. \tag{12}$$

When there are not many requests at time slot t, the green energy of selected node for processing these requests might not be totally used, which can be reserved and utilized together with the new green energy to be generated at time slot $t+1$ (i.e., $H_n(t+1)$), and the same goes for other nodes. So, the amount of green energy in node n at time slot $t+1$ is

$$G_n(t+1) = \min(G_n(t) - \mu_n(t) + H_n(t+1), G_n^{max}), \forall n \in \mathbb{N}, \tag{13}$$

where G_n^{max} is the battery capacity of edge node n.

To reduce the electricity cost, it is reasonable to set different priorities to use green energy and brown energy. Based on the above definitions, the brown energy to be used can be expressed as $e_n(t) - \mu_n(t)$ and the total energy cost is

$$C(t) = \sum_{n \in \mathbb{N}} \alpha(t)(e_n(t) - \mu_n(t)) + \beta(t)\mu_n(t), \tag{14}$$

in which the coefficients α and β can be decided by edge computing operators to indicate the green energy and brown energy prices.

In our experiments, we aim at minimizing the consumption of brown energy. Intuitively, we can preferentially utilize green energy to replace brown energy, hence we set α much larger than β.

2.4 A Joint MILP Formulation

Now, we can formulate this service management and energy scheduling problem into a *mixed-integer linear programming* (MILP) and its objective is to minimize the long-term total energy cost. By rewriting (14), this problem is formulated as the following one:

Cost-Min:

$$\min : \sum_{t \in T} \sum_{n \in \mathbb{N}} \alpha(t)(e_n(t) - \mu_n(t)) + \beta(t)\mu_n(t),$$
$$\text{s.t.} : (1), (6), (7), (10), (9), (11), (12) \text{ and } (13).$$

In fact, when the supply of green energy is sufficient, that is, $\mu_n(t)$ is large enough, the problem can be transformed into minimizing the consumption of green energy.

3 The DRL-Based Edge Computing Energy Scheduling Algorithm

To cope with the computational complexity and the complex network conditions, we propose a DRL-based algorithm to solve the service management and energy scheduling problem in edge computing. As a model-free approach, it can automatically learn from transitions and then give adequate control solutions accordingly at runtime without any prior knowledge of the energy dynamics or network statistics. At the very beginning, the DRL agent has no knowledge of how to make decisions (or take action). It receives the *states* of the network conditions, including service demands, green energy amount, and service location, and uses them as the input of the neural network to produce corresponding decisions of service management and energy scheduling as *actions*. Then, a corresponding *reward* can be calculated based on how satisfactory the decisions are and the weights of neuronal network will be updated accordingly. Theoretically, if the agent experiences enough states, it will be able to make optimal-approaching decisions. So, in the first place, we define the three key elements of RL, i.e., *state* space S, *action* space A, and *reward* r as follows,

S: The state $s \in S$ is represented as an array composed of the indexes of nodes which host the service, available green energy amount, and the arrival rates of requests processing on each edge server at time slot t, i.e.,

$$\begin{aligned} s(t) = &((X_f(t), \forall f \in \mathbb{F}), \\ &(G_n(t), \forall n \in \mathbb{N}), \\ &(\lambda_n(t), \forall n \in \mathbb{N})), \end{aligned} \tag{15}$$

where $X_f(t)$ is the index of the node to host service f at time slot t.

A:To minimize the long-term energy cost, two decisions should be made as

$$\begin{aligned} a(t) = &((X_f(t), \forall f \in \mathbb{F}), \\ &(\mu_{X_f(t)}(t), \forall f \in \mathbb{F})) \end{aligned} \tag{16}$$

to indicate the services migration locations and corresponding energy scheduling. Define $\mathbb{O} = \{0, \frac{1}{O}G_{X_f(t)}(t), \frac{2}{O}G_{X_f(t)}(t), ..., G_{X_f(t)}(t)\}$ as the percent of remained green energy $G_{X_f(t)}t$ in node $X_f(t)$ to be used, where the value of O can be customized by the network operators. The action $\mu_{X_f(t)}(t)$ as the green energy usage can be decided as

$$\mu_{X_f(t)}(t) \in \mathbb{O} \cdot G_{X_f(t)}(t), \forall f \in \mathbb{F}. \tag{17}$$

Note that when $\mu_{X_f(t)}(t) \geq e_{X_f(t)}(t)$, we can only need to use $e_{X_f(t)}(t)$ units green energy, remaining the $(\mu_{X_f(t)}(t) - e_{X_f(t)}(t))$ units for node $X_f(t)$.

r: We set the reward at time slot t as the negative of (14) defined in our **Cost-Min** problem, i.e.,

$$r(t) = -C(t). \tag{18}$$

With the three key elements, the agent can interact with environment and evaluate the policy π on the basis of the action-value function as

$$Q(s(t), a(t)) = r(t) + \gamma Q(s(t+1), \pi(s(t+1))), \tag{19}$$

where $Q(s(t+1), \pi(s(t+1)))$ contains the discounted future potential reward, directing the agent with long-term view. The optimal policy always select the action that can obtain the maximal Q value, so the state-action value function can be updated according to transition $j = (s, a, r, s')$ as

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s', a') - Q(s,a)\right). \tag{20}$$

DQN applies neural network with weights θ to fit Q values so that it can generically evaluate the state-action that has never been seen before. Furthermore, DQN introduces experience replay memory $\mathbb{M}$ and target network with weights θ' to eliminate the correlation of experiences and improve the stability of evaluating future reward. So mini-batch $\mathbb{B}$ consisting of experiences randomly sampled from $\mathbb{M}$ can be learned at once with learning rate α. Accordingly, the loss function is expressed as

$$L(\theta) = \sum_{j \in \mathbb{B}} \left(r_j + \gamma \max_{a'} Q(s'_j, a'; \theta') - Q(s_j, a_j; \theta)\right)^2. \tag{21}$$

3.1 Prioritized Replay Memory Design

The agent needs to learn sufficient transition experiences to before it can make appropriate decisions. However, with all possible service placement and energy scheduling decisions, it is impossible to try all possible actions. Some existing work shows that potentially good actions should be explored with a higher priority to accelerate the training procedure [14]. It is undeniable that the use of prioritisation in sampling transitions from experience replay does play the role of improving training, however, the prioritized transition with high temporal-difference (TD) error may cause the jitter of result during long-term training. We notice that the solution of **Cost-Min** problem in Section 2.2 can also be utilized as a guideline to judge the efficiency of the actions made by the agent when it is not well trained. We define a normalized difference error σ of the instant reward $r(t)$ and the solution of problem **Cost-Min** Y_{CM} as

$$\sigma = \frac{1}{|Y_{CM} - r(t)| + 1}. \tag{22}$$

By integrating the error σ, with the *Temporal-Difference* (TD) error δ as

$$\delta = r + \gamma \max_{a'} Q(s', a'; \theta') - Q(s, a; \theta), \tag{23}$$

we define the priority of transition j consisting of two parts: CM-error $|\sigma|$ related to the **Cost-Min** formulation and the TD error δ calculated by the agent, as

$$p_j = \tau \cdot \sigma + (1 - \tau) \cdot \delta, \tag{24}$$

where τ is the controlling parameter to balance the importance of σ and δ. Initially, we would like to set $\tau = 1$ for the inexperienced DQN. Once it has gained some experiences, we attenuate τ to 0 since we can trust the well trained DQN then. The probability of sampling transition j can be calculated as

$$P_j = \frac{p_j^d}{\sum_{k \in \mathbb{M}} p_k^d}, \tag{25}$$

where the d is the prioritization exponent.

Algorithm 1. Prioritized DRL-based Agent Training

```
1: INPUT: minibatch $\mathbb{B}$, step-size $\eta$, replay period $R$
2:   Randomly initialize neuronal network $Q(s,a|\theta)$ with weights $\theta$ and target network $Q(s,a|\theta')$ with weights $\theta'$
3: Initialize replay memory $M$
4: Receive the initial observed state $s(0)$ and make action $a(0)$
5: for $t = 0$ to $T_{episode}$ do
6:   Generate action $a(t)$ by exploration
7:   Interact with the environment using actions $a(t)$, obtain reward $r(t)$ and new state $s(t+1)$ and store $(s(t), a(t), r(t), s(t+1)) \rightarrow M$ with $p_t = 1$
8:   if $t \equiv 0 \bmod R$ then
9:     for $j = 1 \rightarrow M$ do
10:      Sample transition $j$ according to $P_j = \frac{p_j^b}{\sum_{k \in \mathbb{K}} p_k^b}$
11:      Compute the importance-sampling weight $w_j = \frac{(\mathbb{B} \cdot P_j)^{-\rho}}{\max_{i \in \mathbb{B}} w_i}$
12:      Compute the TD-error $\delta_j$ and CM-error $\sigma$
13:      Update transition priority according to (22) and (24)
14:      Accumulate weight-change $\Delta \leftarrow \Delta + w_j \cdot \sigma_j \cdot \nabla_\theta Q(S_{j-1}, a_{j-1})$
15:    end for
16:    Update neuronal network weights $\theta \leftarrow \theta + \eta \cdot \Delta$ and set $\Delta = 0$
17:    Copy weights to target network every $\eta$ steps $\theta' \leftarrow \theta$
18:  end if
19: end for
```

3.2 Agent Training Algorithm

We summarize our pDRL algorithm in Algorithm 1 with the incorporate of the prioritized replay memory. To start the pDRL training procedure, we first define minibatch $\mathbb{B}$, step-size η, replay period R in line 1. Then, we randomly set the weights of the evaluation network $Q(s,a|\theta)$ as θ, and the weights of target networks $Q(s,a|\theta')$ are cloned from the evaluation network(see line 2). Now, we are ready to start the agent training with initial state $s(0)$ and obtain an action $a(0)$, as shown in line 4. This action $a(0)$ is executed to get a reward $r(0)$ and the resulted in new state $s(1)$. This procedure is repeated and the new transitions $(s(t), a(t), r(t), s(t+1))$ are stored in the replay memory with an initial priority of 1 (line 7), as the newly generated transitions should always be explored first. For every R time, we sample $|\mathbb{B}|$ transitions according to their probability P_j and compute the importance-sampling weight as w_j (see line 11). The priority is also updated as $p_j = \tau \cdot \tau + (1 - \tau) \cdot \delta$ according to current TD-error δ_j and CM-error σ in line 13. After that, as shown in lines 16 and 17, the weights of the evaluation network and the target network are also updated according to the accumulated weight-change calculated in line 14. The above procedure iterates until convergence or reaching the predefined episode bound. Once the agent gets well trained, it shall be already to make the right decision towards energy cost minimization according to any real-time observation (i.e., states).

TABLE 2
Default Settings

Parameters	Values
The cost of processing a request	0.2
The cost of migrating a vm between a hop	15
The cost of one request's communication between a hop	0.1
The weight of green energy cost in reward	1
The weight of brown energy cost in reward	1000
The number of replaced size per iteration	512
The size of prioritized replay memory	4000
The number of fully connected layers	3
The learning rate	0.0001
The discount rate	0.9

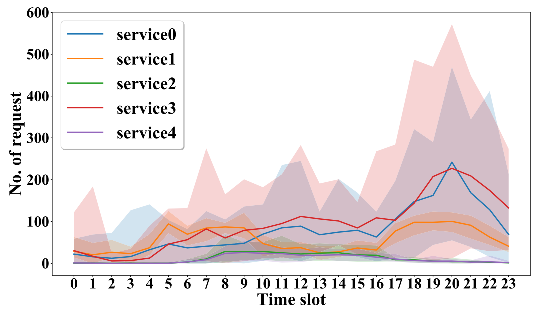
4 Performance Evaluation

To validate the efficiency of our prioritized DRL-based algorithm, we conduct extensive simulations and report the results in this section. The simulated environment and all algorithms are implemented by Python 3.7. The neural networks involved in DRL-based algorithms are implemented using Tensorflow 1.9 framework and the optimization problem of CM is solved using Gurobi 9.0.2. All the experiments are conducted on a server equipped with a 1.80GHz 8-Core Intel Core CPU i7-8550U processor. Our pDRL consists of two networks, namely *eval_net* and *target_net*, respectively. Both *eval_net* and *target_net* have the same network structure. The input size is the dimension of state, and the input layer is connected to three fully connected layers called *n_l1*, *n_l2*, and *n_l3*. For the first layers *n_l1*, the size is 236, and the size of *n_l2* is 118 while *n_l3* is 59. The activation function is *relu* and the output size is the numbers of edge nodes. The parameters of *target_net* will be updated by the *eval_net* during the agent training. The default settings are summarized in Table 2.

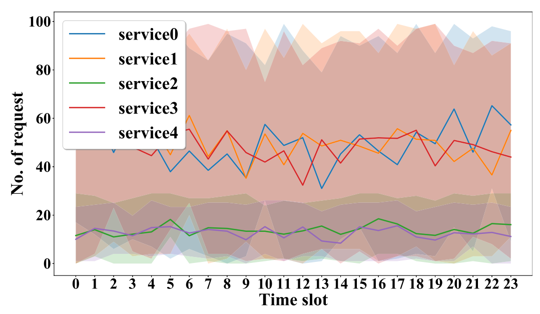
4.1 Simulation Result

We consider two well-known network topologies, i.e., ARPA Network and NSF Network, consisting of 9 and 14 edge servers, respectively. Each server is associated with an amount of requests and green energy at each time slot. We set 5 different types of service from real world, including Uber request, public work request, food order request, and base station request. As shown in Fig. 2a, these requests follow different distribution patterns. The solid line indicates the average amount of one service request arriving in all the edge nodes, and the shaded area is the variation range of request amount. As we can see, different service requests have different amounts and trends throughout the day. The green energy generation trace is extracted from ENTSOE[1], including solar energy and wind

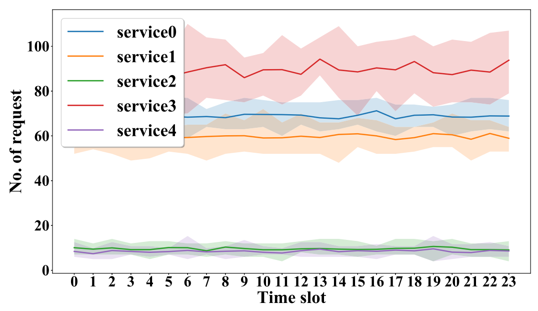
1. https://transparency.entsoe.eu/dashboard/show



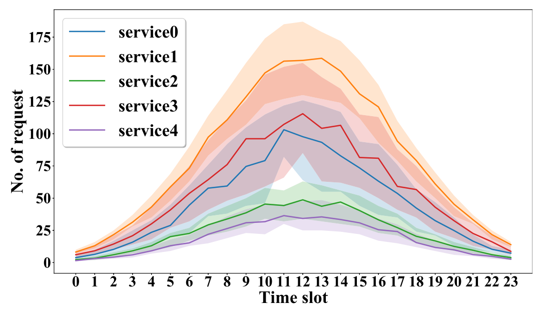
(a) Real World Request Pattern



(b) Uniform Pattern



(c) Normal Pattern



(d) Time-related Pattern

Fig. 2. Different request patterns.

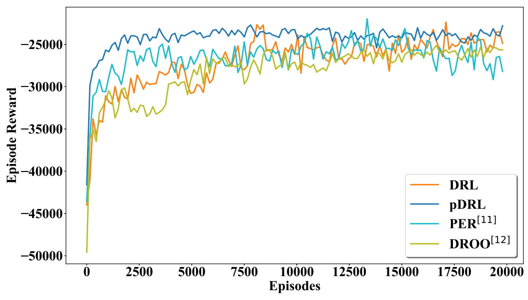
energy. To be practical, we set the time slots from 17 to 23 as the peak hours and the electricity prices in this period are tripled.

The communication cost and migration cost are set according to the hops between any two edge servers, and different services have different unit cost. We evaluate the performance of traditional model-based algorithm by minimizing the instant cost of each time slot in (14) ("**Instant**"), greedy algorithm by choosing the node with max green energy remained ("**Greedy**"), DRL-based algorithm ("**DRL**"), Deep Reinforcement Learning for Online Computation Offloading algorithm ("**DROO**"), Prioritized Experience Replay DQN ("**PER**"), and our prioritized DRL-based algorithm ("**pDRL**").

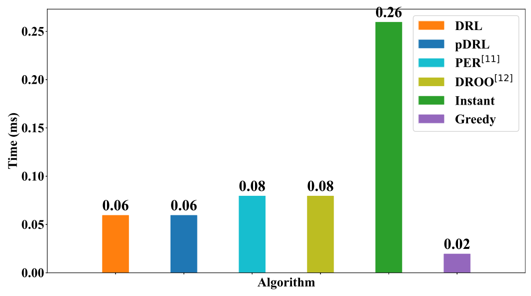
First, Figs. 3a and 4a shows the convergence trend of episode rewards obtained by our pDRL, DRL, PER, and DROO algorithms. Taking NSF network as an example, we have trained these neural networks with 40000 episodes. With the increasing number of training episodes, both DRL and pDRL eventually converge. It can be seen that our pDRL converges much faster than other DRL, converging in the 13000th episodes. Moreover, our pDRL reaches a lower cost than that of other DRL algorithms, as 17761 for pDRL. This is because our pDRL prioritizes the transitions according to its reward, and the good transitions can be learned more effectively, leading to faster convergence and lower cost. Fig. 3a shows the same trend on ARPA Network, validating the effectiveness of our pDRL on different network topologies.

Decision-making time is a critical metrics, determining the availability of an algorithm. Hence, we present the average execution time of four algorithms on two different network topologies. As shown in Figs. 4b and 3b, all DRL-based algorithms use neural networks to make the placement decisions and can complete decisions in a short time, i.e., DRL and our pDRL spend 0.06ms, 0.07ms on both NSF and ARPA Network, respectively, while PER and DROO take 0.08ms, 0.08ms on the above two networks. The Greedy algorithm, primarily sorting the node according to its remained green energy, is $O((|\mathbb{F}| + |\mathbb{N}|) \cdot log|\mathbb{N}|)$ of time complexity, increasing almost linearly with the service number and edge node number. It can be seen that the execution time on ARPA network is 0.02ms in Fig. 3b and NSF is 0.02ms in 4b, respectively. While the Instant algorithm is executed by Gurobi's ILP solver, which solves this problem by heuristically exploring all the solution trees to obtain the optimal one. Consequently, its time complexity is almost $O(|\mathbb{N}|^{|\mathbb{F}|})$, increasing exponentially with the increase of service number or edge node number. As a result, it need more 0.17ms when the network topology expands from 9 nodes to 14 nodes. But in DRL-based algorithms, the results show no obvious execution time increase with the network scale, because the execution time of neural network is only related to its structure. Hence they can be applied to large scale networks and provide a fast response.

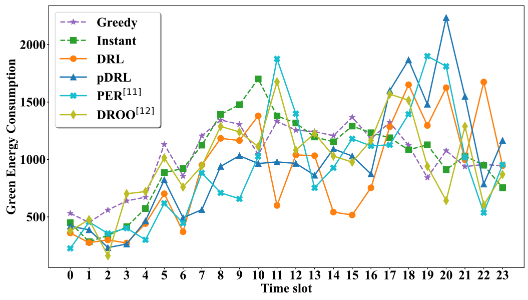
Then, we show the green energy consumption and total cost of the four algorithms. Figs. 3c and 4c show the green energy consumptions of 24 time slots, and Figs. 3d and 4d show the costs. In our experiments, brown energy price is tripled during peak hours, i.e., time slots from 17 to 23, comparing to off-peak hours, i.e., time slots from 0 to 16. When there is insufficient green energy, Greedy and Instant algorithms myopically focus on optimizing the on-site energy cost based on present network state without any consideration of the future probabilities. So, they intend to maximize the green energy utility at each off-peak time slot as shown in Figs. 3c and 4c. When peak hour with higher electricity price falls, there is not sufficient green energy available and more brown energy is needed, leading to large cost as shown in Figs. 3d and 4d. The total cost of Greedy and Instant is 36224 and 27529 respectively. As for DRL-based algorithms, they can learn from past experiences and evaluate probabilities of the future electricity price changing. With this knowledge, they decide to use fewer green energy in time slots



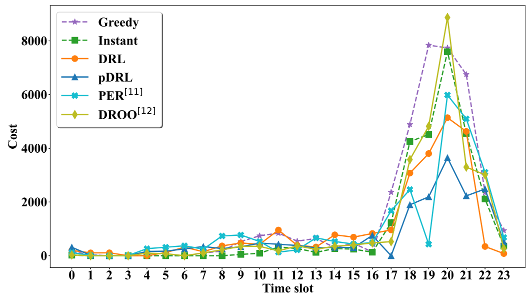
(a) Training Requests



(b) Execution Time



(c) Green Energy Consumption

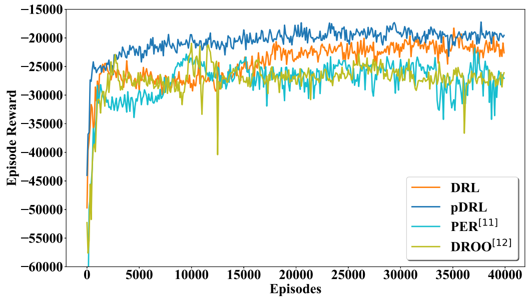


(d) Cost

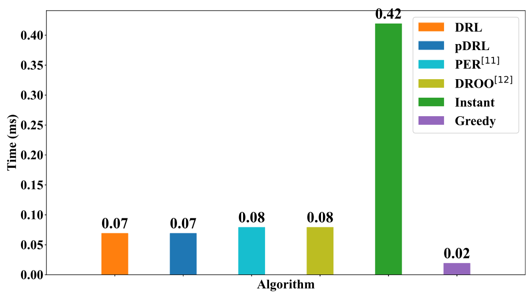
Fig. 3. Simulation results on ARPA network.

$0 \sim 16$, and reserve part of green energy for time slots $17 \sim 23$ with a higher electricity price, as shown in Figs. 3c and 4c. As a result, we can use the reserved green energy during peak hours, and the brown energy cost of pDRL and oher DRL-based algorithms are much lower than Greedy and Instant, especially in time slots $17 \sim 22$. Furthermore, compared with other DRL-based algorithms, our pDRL has an advanced prioritized replay memory to learn better transitions, and consequently manages the service and schedules the energy more efficiently. The total cost of pDRL is further reduced in Figs. 3d and 4d, compared with other DRL-based algorithms. So we can conclude that our pDRL algorithm deals with the environmental dynamics and shows its advantages under this real-world trace.

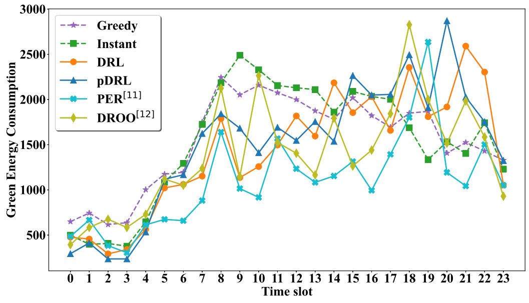
More generally, we evaluate our pDRL with 3 different types of service request patterns, as shown in Figs. 2b, 2c, and 2d. Similar to Fig. 2a, the solid line indicates the average amount of service request, and the shaded area represents the amount varying range. The request pattern in Fig. 2b follows a uniform distribution, and each service has different upper and lower bounds. The request pattern in Fig. 2c follows a normal



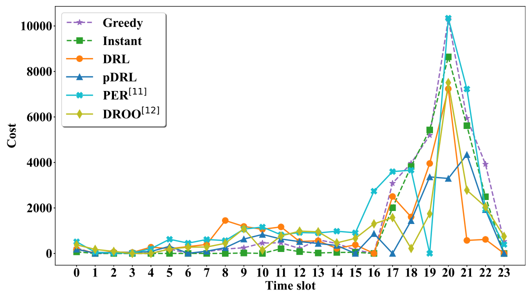
(a) Training Requests



(b) Execution Time



(c) Green Energy Consumption



(d) Cost

Fig. 4. Simulation results on NSF network.

TABLE 3
Green Energy Consumptions Under Different Request Patterns

Algorithm \ Pattern	real world	uniform	normal	time dependent
Greedy	36898	45266	48373	50245
Instant	36746	44935	48514	49305
DRL	34216	33460	41306	43687
PER[14]	32473	33999	34459	37684
DROO[15]	33188	34903	40035	44330
pDRL	35660	39010	43715	46800

TABLE 4
Costs Under Different Request Patterns

Algorithm \ Pattern	real world	uniform	normal	time dependent
Greedy	36224	32644	46062	40885
Instant	27529	16208	32223	30174
DRL	25325	39068	48936	48421
PER[14]	28259	32662	52711	54460
DROO[15]	24829	46738	44402	47344
pDRL	21638	30139	44360	41784

distribution, and each service has a different mean and variance. While the number of service request in Fig. 2d varies with time, presenting a Gaussian distribution in the time dimension, and a certain range of random fluctuations are carried out near the Gaussian distribution value. The performance of four algorithms on the NSF network are listed in Tables 3 and 4.

Table 3 shows the total green energy consumptions of four algorithms. We can find that the green energy consumption of pDRL is always higher than other DRL-based algorithms. This is because with the prioritized replay memory design, our pDRL can learn better than DRL within the same training episodes, and hence can find better service location and make better energy scheduling solution. It is also noticeable that Greedy and Instant consume more green energy than four DRL-based algorithms. The reason is that they always try to migrate services to the nodes with more green energy and use as much green energy as possible in each time slot. While frequent migration results in more energy consumption, especially more brown energy cost when green energy is insufficient, as shown in Table 4. Table 4 also shows that the cost of pDRL always outperforms other algorithms, followed by DRL. Since Instant myopically finds a single-time-slot optimal solution and is second to DRL-based algorithms, Greedy always shows the worst performance.

Averagely, our pDRL accelerates the convergence speed by 37.67% and reduces the long-term OPEX by 32.70%, compared with state-of-the-art studies.. The training acceleration is because our pDRL prioritizes the transitions according to its reward. Moreover, the solution of Cost-Min problem in Section 2.2 is used as a guideline to judge the quality of the actions. As a result, the good transitions can be learned more effectively by the agent and the training procedure is accelerated. As for the lower cost, it can be seen from Figs. 3c and 4c that our pDRL saves the green energy from time slot 0 to 16 with lower price for future use while Greedy and Instant algorithms myopically focus on optimizing the on-site energy cost based on present network state without any consideration of the future probabilities. For example, at time slot 7, the green energy consumption of Greedy and Instant in ARPA network is 1206 and 1125 respectively, while our pDRL is 563. With such long term energy scheduling, during the peak hour with higher electricity price, our pDRL saves sufficient green energy and consumes less brown energy, leading to lower cost compared with Greedy and Instant solutions.

5 Related Work

5.1 A Review on Edge Energy Management

With the ever-growing computing demand, energy consumption of edge computing has been becoming a notable problem. It is important for edge servers to properly manage energy while scheduling resource. For example, [16] studies how to balance *mobile edge computing* (MEC) system performance and energy consumption by properly migrating services between edge servers with request delay constraint. Zhang et al. [17] leverage DVFS to adjust the power consumption of the VMs, with certain service failure probability. Guo et al. [18] propose an efficient request offloading and resource allocation strategy to reduce energy consumption and application completion time on smart mobile devices. Chen et al. [19] focus on energy-efficient offloading in MEC and propose a novel algorithm to reduce total energy consumption. Guo et al. [20] study how to adjust the coverage range and allocate channel resources of base stations with EH powering, to reduce the brown energy cost. Xiong et al. [21] focus on a wireless energy and data transfer supported UAV system and propose a novel DRL-based approach for long-term utility of UAV.

For energy-constrained edge devices, offloading requests to the nearby edge servers is a feasible solution to prolong device lifetime. Sun et al. [22] focus on the optimization of device scheduling problem with the limitation of energy and devise an energy-aware dynamic scheduling algorithm. Xu et al. [23] investigate request offloading of AI applications in MEC with the consideration of energy scheduling and propose an energy-aware strategy. Yi et al. [24] consider how to divide the devices into multiple edge sub-networks where a MEC using green powered server is equipped, to minimize the operation cost with limited green energy. Geng et al. [25] leverage the Big. Little architecture of the multicore based edge devices to jointly make request offloading and local scheduling decisions, which schedules requests to the appropriate CPU core according to request priority, to reduce the energy consumption while satisfying the completion time constraints.

Leveraging energy harvesting technology is another viable way to lengthen the device lifetime. For example, [26] study the trade-off between request latency and energy consumption in MEC, and propose different request scheduling methods. However, most of the existing solutions for energy management are model-based with simplified network or assumptions on environment dynamics, besides, they mainly target on one-shot optimization, which may fail to adopted well in dynamically changing real scenarios.

5.2 A Review on RL in Edge Computing

Reinforcement learning, as a model-free approach without any prior knowledge, can automatically learn the dynamics and make appropriate decisions accordingly at runtime.

Many existing works leverage different RL-based algorithms in edge caching. Qian et al. [27] propose a content push and cache divide-and-conquer strategy based on RL, where Q-learning decides which file to be cached and DQN decides which BS to cache content, and this framework effectively learns the content popularity and predicts the future user demands. Wang et al. [28] propose a DQN-based algorithm with attention-weighted federated learning, which selects the cache nodes and the replacement of cache contents in mobile networks, with the goal of improving the cache hit rate. Zheng et al. [29] cast edge caching problem into a MINLP problem and propose a RL-based method to solve the problem with consideration of minimizing the total energy consumption.

Request/task offloading and resource allocation are also widely studied in edge computing, and DRL is a particularly effective strategy to tackle this complex controlling problem. For example, Chen et al. [30] utilize DRL to jointly decide energy-efficient request offloading and computing resource allocation, which achieves a great reduction on energy consumption compared with traditional algorithms. Liu et al. [31] use DRL to solve the problem of load balancing with the consideration of communication and computation requirements. With the goal of achieving close-to-optimal performance, [32], [33] solve complex request offloading problems based on RL under various constraints, all of them achieved better performance than traditional heuristic or convex optimization algorithms. Taking [34] as an example, the authors propose a novel deep Q-learning-based system to solve the problem of resource provisioning and task scheduling with the goal of improving energy cost efficiency and fast convergence. RL is also used to solve traditional ICT system management. Liu et al. [35] study the problem of sub-network division in the IoT and leverage DQN to deal with the dynamic load balancing of edge servers. Kim et al. [36] design AutoScale based on reinforcement learning algorithm to decide running inference on the CPU or co-processors so as to improve energy efficiency in mobile systems.

RL-based solutions have shown their great improvement in edge computing resource management that is difficult for traditional model-based methods. This inspires us to introduce RL to the field of energy scheduling and service management in edge computing. However, taking advantage of RL into this field is limited to the effiency of trial-and-error, how to utilize the vast amount of samplings required for agent training with effect in energy scheduling and service management is our goal.

6 Conclusion

In this paper, we investigate the problem of service management and energy scheduling in edge computing. To minimize the long-term energy cost, every decision should be made carefully in a foresighted way by taking current network statics and future vision into account. It is hard to use the traditional model-based algorithm to address such problem. Instead, we resort to the newly proposed DRL technique, i.e., DQN, to design a model-free solution. To accelerate the DQN training converge, we further customize it by leveraging the traditional model-based solution as a guideline to update the transition sample priority. Different with Prioritized Replay Memory based on Temporal-Difference Error, pDRL utilized the episode reward of transition sample as the difference error to adapt to the sparsity of transition sample with high reward in real scenarios. Extensive experiments verify the correctness of our design and the efficiency of our algorithm by the fact that it outperforms traditional model-based method algorithm. Our study indicates that DRL provides the possibility of escaping from the model-based solution with assumptions. A trained agent can take a picture of the whole network as input to make control decisions according to the desired objective for complex network control problems. While, we shall not only simply apply the DRL algorithm, but shall also well customize it according to the characteristics of the control problem. In future work, several approaches could be extended from this work. For example, the unused green energy can be transferred to neighbour servers to improve energy efficiency. Moreover, pDRL could be exploited in mobile edge computing scenarios assisted by multiple agents to support distributed service management and energy scheduling in large scale networks.

References

[1] M. Lemay, K. K. Nguyen, B. S. Arnaud, and M. Cheriet, "Toward a zero-carbon network: Converging cloud computing and network virtualization," *IEEE Internet Comput.*, vol. 16, no. 6, pp. 51–59, Nov./Dec. 2012.

[2] P. Steenhof et al., "A protocol for quantifying the carbon reductions achieved through the provision of low or zero carbon ICT services," *Sustain. Comput.: Inform. Syst.*, pp. 23–32, 2012.

[3] A. A. Chien, R. Wolski, and F. Yang, "Zero-carbon cloud: A volatile resource for high-performance computing," in *Proc. IEEE Int. Conf. Comput. Informat. Technol.*, 2015, pp. 1997–2001.

[4] F. Yang and A. A. Chien, "Zccloud: Exploring wasted green power for high-performance computing," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2016, pp. 1051–1060.

[5] G. Zhang, W. Zhang, Y. Cao, D. Li, and L. Wang, "Energy-delay tradeoff for dynamic offloading in mobile-edge computing system with energy harvesting devices," *IEEE Trans. Ind. Informat.*, vol. 14, no. 10, pp. 4642–4655, Oct. 2018.

[6] J. Mineraud, L. Wang, S. Balasubramaniam, and J. Kangasharju, "Hybrid renewable energy routing for isp networks," in *Proc. IEEE Int. Conf. Comput. Commun.*, 2016, pp. 1–9.

[7] F. Guo, L. Ma, H. Zhang, H. Ji, and X. Li, "Joint load management and resource allocation in the energy harvesting powered small cell networks with mobile edge computing," in *Proc. IEEE Conf. Comput. Commun. Workshops*, 2018, pp. 299–304.

[8] S. Conti, G. Faraci, R. Nicolosi, S. A. Rizzo, and G. Schembra, "Battery management in a green fog-computing node: A reinforcement-learning approach," *IEEE Access*, vol. 5, pp. 21 126–21 138, 2017.

[9] Y. Yang, D. Wang, D. Pan, and M. Xu, "Wind blows, traffic flows: Green internet routing under renewable energy," in *Proc. IEEE Conf. Comput. Commun.*, 2016, pp. 1–9.

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

[11] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1619–1632, Aug. 2018.

[12] D. Silver et al., "Mastering the game of Go without human knowledge," *Nature*, vol. 550, pp. 354–359, 2017.

[13] Z. Xiong, Y. Zhang, D. Niyato, R. Deng, P. Wang, and L. Wang, "Deep reinforcement learning for mobile 5G and beyond: Fundamentals, applications, and challenges," *IEEE Veh. Technol. Mag.*, vol. 14, no. 2, pp. 44–52, Jun. 2019.

[14] I. A. T. Schaul, J. Quan, and D. Silver, "Prioritized experience replay," in *Proc. Int. Conf. Learn. Representations (Poster)*, 2015.

[15] L. Huang, S. Bi, and Y.-J. A. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Trans. Mobile Comput.*, vol. 19, no. 11, pp. 2581–2593, Nov. 2020.

[16] H. Badri, T. Bahreini, D. Grosu, and K. Yang, "Energy-aware application placement in mobile edge computing: A stochastic optimization approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 4, pp. 909–922, Apr. 2020.

[17] W. Zhang, Z. Zhang, S. Zeadally, H. Chao, and V. C. M. Leung, "Energy-efficient workload allocation and computation resource configuration in distributed cloud/edge computing systems with stochastic workloads," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1118–1132, Jun. 2020.

[18] S. Guo, J. Liu, Y. Yang, B. Xiao, and Z. Li, "Energy-efficient dynamic computation offloading and cooperative task scheduling in mobile cloud computing," *IEEE Trans. Mobile Comput.*, vol. 18, no. 2, pp. 319–333, Feb. 2019.

[19] X. Chen, J. Zhang, B. Lin, Z. Chen, K. Wolter, and G. Min, "Energy-efficient offloading for DNN-based smart IoT systems in cloud-edge environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 3, pp. 683–697, Mar. 2022.

[20] F. Guo, L. Ma, H. Zhang, H. Ji, and X. Li, "Joint load management and resource allocation in the energy harvesting powered small cell networks with mobile edge computing," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 299–304.

[21] Z. Xiong et al., "UAV-assisted wireless energy and data transfer with deep reinforcement learning," *IEEE Trans. Cogn. Commun. Netw.*, vol. 7, no. 1, pp. 85–99, Mar. 2021.

[22] Y. Sun, S. Zhou, Z. Niu, and D. Gündüz, "Dynamic scheduling for over-the-air federated edge learning with energy constraints," *IEEE J. Sel. Areas Commun.*, vol. 40, no. 1, pp. 227–242, Jan. 2022.

[23] Z. Xu et al., "Energy-aware inference offloading for DNN-driven applications in mobile edge clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 4, pp. 799–814, Apr. 2021.

[24] Y. Liu, S. Xie, Q. Yang, and Y. Zhang, "Joint computation offloading and demand response management in mobile edge network with renewable energy sources," *IEEE Trans. Veh. Technol*, vol. 69, no. 12, pp. 15 720–15 730, Dec. 2020.

[25] Y. Geng, Y. Yang, and G. Cao, "Energy-efficient computation offloading for multicore-based mobile devices," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 46–54.

[26] J. Luo et al., "Container-based fog computing architecture and energy-balancing scheduling algorithm for energy IoT," *Future Gener. Comput. Syst.*, vol. 97, pp. 50–60, 2019.

[27] Y. Qian, R. Wang, J. Wu, B. Tan, and H. Ren, "Reinforcement learning-based optimal computing and caching in mobile edge network," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 10, pp. 2343–2355, Oct. 2020.

[28] X. Wang, R. Li, C. Wang, X. Li, T. Taleb, and V. C. M. Leung, "Attention-weighted federated deep reinforcement learning for device-to-device assisted heterogeneous collaborative edge caching," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 1, pp. 154–169, Jan. 2021.

[29] H. Zheng, H. Zhou, N. Wang, P. Chen, and S. Xu, "Reinforcement learning for energy-efficient edge caching in mobile edge networks," in *Proc. IEEE Conf. Comput. Commun. Workshops*, 2021, pp. 1–6.

[30] X. Chen and G. Liu, "Energy-efficient task offloading and resource allocation via deep reinforcement learning for augmented reality in mobile edge networks," *IEEE Internet Things J.*, vol. 8, no. 13, pp. 10 843–10 856, Jul. 2021.

[31] Q. Liu, T. Xia, L. Cheng, M. van Eijk, T. Ozcelebi, and Y. Mao, "Deep reinforcement learning for load-balancing aware network control in IoT edge systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 6, pp. 1491–1502, Jun. 2022.

[32] S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou, and X. Shen, "Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach," *IEEE Trans. Mobile Comput.*, vol. 20, no. 3, pp. 939–951, Mar. 2021.

[33] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, "Fast adaptive task offloading in edge computing based on meta reinforcement learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 1, pp. 242–253, Jan. 2021.

[34] M. Cheng, J. Li, and S. Nazarian, "Drl-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers," in *Proc. 23rd Asia South Pacific Des. Automat. Conf.*, 2018, pp. 129–134.

[35] Q. Liu, L. Cheng, T. Ozcelebi, J. Murphy, and J. Lukkien, "Deep reinforcement learning for IoT network dynamic clustering in edge computing," in *Proc. IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput.*, 2019, pp. 600–603.

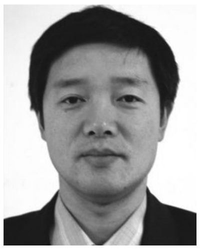
[36] Y. G. Kim and C.-J. Wu, "Autoscale: Energy efficiency optimization for stochastic edge inference using reinforcement learning," in *Proc. Annu. IEEE/ACM Int. Symp. Microarchit.*, 2020, pp. 1082–1096.

Lin Gu (Member, IEEE) received the MS and PhD degrees in computer science from the University of Aizu, Fukushima, Japan in 2011 and 2015. She is currently an associate professor with the School of Computer Science and Technology, Huazhong University of Science and Technology, China. Her current research interests include serverless computing, network function virtualization, cloud computing, software-defined networking, and data center networking. She has authored 2 books and more than 40 papers in refereed journals and conferences in these areas. She is a senior number of CCF.

Weiying Zhang received the graduation degree from the Huazhong University of Science and Technology, the BE degree from Northeastern University in 2019, and the MS degree in 2022. Her current interests mainly focus on the service management and energy scheduling.

Zhongkui Wang received the graduation degree from the Huazhong University of Science and Technology, the BE degree from Jilin University in 2018, and the MS degree in 2021. His interests mainly focus on the computing resource management and reinforcement learning.

Deze Zeng (Member, IEEE) is currently a full professor with the School of Computer Science, China University of Geosciences, Wuhan, China. His current research interests mainly focus on edge computing, cloud computing, and IoT. He has authored 3 books and more than 120 papers in refereed journals and conferences in these areas. He also received 5 best paper awards from IEEE/ACM conferences and journals. He serves in editorial boards of *IEEE Transactions on Sustainable Computing*, *Journal of Network and Computer Applications*, *Frontiers of Computer Science*, and *Open Journal of Computer Society*. He has been in the organization or program committees of many international conferences. He is a senior member of CCF.



Hai Jin (Fellow, IEEE) received the PhD degree in computer engineering from HUST in 1994. He is a chair professor of computer science and engineering with the Huazhong University of Science and Technology (HUST) in China. In 1996, he was awarded a German Academic Exchange Service fellowship to visit the Technical University of Chemnitz in Germany. He worked with The University of Hong Kong between 1998 and 2000, and as a visiting scholar with the University of Southern California between 1999 and 2000. He was awarded Excellent Youth Award from the National Science Foundation of China in 2001. He is a Fellow of CCF, and a life member of the ACM. He has co-authored more than 20 books and published more than 900 research papers. His research interests include computer architecture, parallel and distributed computing, big data processing, data storage, and system security.