Peer-to-Peer Networking and Applications (2022) 15:2619–2632
https://doi.org/10.1007/s12083-022-01369-6

Intelligent edge content caching: A deep recurrent reinforcement learning method

Haitao Xu[1] · Yuejun Sun[1] · Jingnan Gao[1] · Jianbo Guo[2]

Received: 27 November 2021 / Accepted: 10 August 2022 / Published online: 6 September 2022

Abstract
With the rise of 5G network and the rapid growth of user equipment, there exists a gap between the stringent requirements of emerging applications and the actual functionality of the Internet. In particular, transmitting data over long network links imposes high costs, which can be addressed by the edge caching (EC) method. EC caches the content at the edge server to avoid the extraordinary cost of backhaul link communication. However, in existing EC efforts, it is common to assume either known content popularity or a two-phase caching that is predicted content popularity prior to the caching action, the former being less feasible and the latter increasing the cost of deployment to the real world. A caching strategy is proposed in this paper to cope with this problem that can be feasible end-to-end deployed and has a lower caching cost. Specifically, we first investigate the system cost, including network communication cost, cache over storage cost, and cache replacement cost. And we model the EC problem as a Markov Decision Process (MDP). Then, the Double Deep Recurrent Q Network (DDRQN) algorithm is studied to solve the EC-based MDP problem. Finally, compared with other intelligent caching strategies, the proposed caching strategy can improve the system reward by up to 24% and the cache hit rate by up to 22% under certain conditions.

Keywords Edge computing · Edge caching · Edge intelligence · Deep recurrent reinforcement learning

1 Introduction

With the vigorous development of 5G network technology, Internet access equipment and network traffic are growing rapidly. The number of devices connected to IP networks will be more than three times the global population in 2023, reaching 29.3 billion, which is more than 1.59 times that of 2018 [1]. However, most connected devices have limited communication and storage resources and finite processing capabilities, which show the mismatch between the stringent requirements for emerging applications and the actual device capabilities [2]. One solution is cloud computing [3], which performs task calculation and content caching in remote cloud data centers. Nonetheless, transmitting large amounts of content over the Wide Area Network (WAN) may bring daunting cost and transmission delay [4].

The development of edge computing brings a new strategy for reducing network costs. Different from cloud computing, edge computing allows wireless devices to offload computation resources to edge servers [5], and provides computing and content caching services using edge servers that are closer to users. With properties closer-to-user, edge computing addresses congestion and high latency on the backhaul link and provides low latency and high-quality services [6]. When users request content, they will first retrieve the cached content of the edge server. Therefore, using a portion of the storage space on the edge server to cache content, especially popular content, will significantly reduce the network delay and congestion.

The edge caching (EC) method has attracted the great interest of many scholars. Xia et al. [7] proved that EC is an NP-complete problem and suggested Lyapunov optimization to minimizing the system costs, including data caching cost, data migration cost, and Quality-of-Service (QoS) penalty. Based on Information-Centric Networking (ICN), Nour et al. [8] proposed a distributed cache

✉ Yuejun Sun
sunyuejun@hdu.edu.cn

Haitao Xu
xuhaitao@hdu.edu.cn

[1] Hangzhou Dianzi University, Hangzhou, Zhejiang, China

[2] Hangzhou Hexing Electrical Co., Ltd, Hangzhou, China

placement scheme to push the popular content down to the next-hop network route, and the unpopular content up to the next-hop network route, bringing popular content closer to the user.

Many researchers have used Deep Learning to solve EC problems with considerable results. To challenge the heavily relies of proactive EC on the accuracy of content popularity prediction, a Bidirectional Deep Recurrent Neural Network (BRNN) scheme was proposed to predict the content requests and update edge cache in [9]. Similarly, Wu et al. [10] designed a Deep Long Short-Term Memory (LSTM) Auto-Encoder algorithm to capture the temporal structures of content. Additionally, due to the importance of the user behavior in EC, Zeng et al. [11] considered a user-centric behavior based contextual zoom algorithm to quickly learn the changes in content popularity and discussed a Modified single-layer nonlinear Convolutional Neural Networks (MCNN) method to mine the relationship between user groups. Take both user features and content features into account, Rathore et al. [12] used auto-encoder and Stacked Denoising Autoencoders respectively to extract the features to estimate content popularity. Nevertheless, these approaches barely take into account the current state of the edge server (e.g., current cached content, energy cost), which lacks flexibility.

The combination of Reinforcement Learning and Deep Learning, called Deep Reinforcement Learning (DRL) [13], is suitable for dealing with complex non-convex optimization problems and has aroused new research interest in recent years. In this case, Zhong et al. [14] first present a DRL framework to solve the content caching problem. With the goal of Quality of Experience (QoE), He et al. [15] summarized the caching model and introduced an adaptive learning rate DRL method to enhance QoE. Besides, Qiao et al. [16] modeled the cooperative caching problem as a double time-scale Markov decision process (DTS-MDP), and a Deep Deterministic Policy Gradient (DDPG)-based cooperative caching scheme was proposed to overcome with high-dimensional state space and continuous-valued action space. Moreover, in [17], Hou et al. exercised LSTM to predict the mobility of vehicles first, and then the result of LSTM was used as the input state of DRL to learn caching policy. Coincidentally, Li et al. [18] employed Gated Recurrent Unit (GRU) algorithm to predict the task popularity first, and used multi-agent DQN to solve the caching problem based on the predicted popularity.

In the above work, they consider the whole system state, however, content popularity is assumed to be either known as the last time slot content popularity as presented in [15, 16], or estimated before choosing the caching actions as proposed in [17, 18] rather than end-to-end policy. Since the content popularity is time-varying, the first assumption is less feasibility, and the second assumption is troublesome to deploy in the real world.

Thus, to address those issues, the end-to-end Double Deep Recurrent Q Network (DDRQN) algorithm is proposed in this paper. In this case, the agent of the DDRQN algorithm observes multi-step states (historical and current states) and uses the GRU network [19] to capture the content popularity for the next period. Besides, the EC system costs, including network communication cost, cache over storage cost, and cache replacement cost are considered to find system-level optimization. Then, the EC problem is modeled as MDP. The DDRQN algorithm can intelligently make caching decisions in the dynamic EC environment to minimize the long-term system-level cost. In summary, the contributions of this paper are as follows:

1. We investigate the cost of the EC system, considering not only network communication cost but also cache over storage cost and cache replacement cost.
2. We propose an intelligent edge caching strategy, which can not only be feasible end-to-end deployed but also has a lower caching cost. And the DDRQN algorithm is designed to minimize system cost.
3. For the first time, we cope with EC problems using GRU based DRL algorithm. In particular, the use of GRU cell in DRL agent can improve the effectiveness of learning time-varying content popularity from the historical and current environments.

The paper is organized as follows. Section 3 summarizes related work. Section 3 introduces the EC system model including system architecture and system cost. Section 4 proposes the solution of the DDRQN algorithm. Section 5 presents the simulation results. Section 6 concludes the paper.

2 Related work

In recent years, EC has attracted great interest from many researchers. In general, most researchers focus on improving key indicators such as hit rate, transmission delay, energy consumption, backhaul load, QoE, QoS, and system cost. And researchers often improve these key indicators by establishing different mathematical models and proposing different novel algorithms.

Based on the heuristic algorithm, Liu et al. [20] modeled the EC problem as an integer programming from the service provider's perspective. And they proposed an extended Page-Hinckley-Test algorithm to reduce network delay and improve energy efficiency of mobile devices. Besides, Hu et al. [21] pointed out that proactive caching can effectively handle redundant traffic loads in future wireless networks.

And they proposed to use game theory to deal with proactive caching in the network. Specifically, they discuss the two main types of caching in proactive caching, namely centralized wireless network caching and distributed wireless network caching respectively. The centralized wireless network caching includes Small Cell Base Station (SBS) caching and Road Side Unit (RSU) caching, and the distributed wireless network caching includes Device-to-Device (D2D) caching and Vehicle-to-Vehicle (V2V) caching. Finally, they used different game theory algorithms for these different wireless proactive caching models. Moreover, Yu et al. [22] proposed a content caching strategy based on Mobility Prediction and Joint User Prefetching (MPJUP). The caching policy first predicts the user's mobility and then prefetches data at the edge server in advance based on the user's mobility and QoE. In addition, they also investigate the problem of collaborative prefetching of data between users and edge servers, which further reduces the backhaul load and improves the network quality. Heuristic-based caching strategy finds approximate solutions in a reasonable time through iteration, but do not guarantee optimal solutions. And the heuristic-based caching strategy needs to formulate the rules of iteration in advance. These conditions limit the application of heuristic algorithms to EC problem to a certain extent.

Based on the deep learning, Saputra et al. [23] proposed centralized EC algorithm and distributed EC algorithm. In the centralized EC algorithm, the cloud server collects information from all edge servers and uses a neural network model to predict the demand for the entire network. In the distributed EC algorithm, each edge server trains the neural network individually and the cloud server only aggregates and updates the model parameters, which ensures privacy and reduces the network communication overhead. Combining Social Content-Centric Networking (SCCN) and edge computing, Liang et al. [24] used Multi-Head Attention based Encoder-Decoder (MAPP) to predict content popularity. MAPP is able to fully consider multiple popularity-related properties. And the experiment results showed that MAPP can achieve better cache hit rate in social data simulation experiments generated using sonetor [25]. Considering both user mobility and user interest, Tang and Kang [26] proposed a machine learning-based intelligent EC strategy. Specifically, they first used LSTM to predict user mobility, and then used Gradient Boosting Decision Tree (GBDT) to predict content popularity. Deep learning-based caching strategy tend to consider more about reducing model loss and improving model accuracy, and assume that higher prediction accuracy leads to better caching performance. However, these strategies separate model learning and cache replacement strategy, and pure model accuracy is not suitable for dynamic edge caching environment [27].

Based on the reinforcement learning, Wang et al. [28] compared the Centralized Double Deep Q Network (C-DDQN) algorithm with the Federated Learning-based Double Deep Q Network (FL-DDQN) algorithm. Experiments showed that C-DDQN tended to have better performance, but FL-DDQN can reduce the communication consumption. By using multiple models, Zhang et al. [29] first used a Grouped Linear Model (GLM) to predict content popularity and then used the predictions as the reinforcement learning state to learn caching strategy. Considering energy constraints, Tang et al. [30] applied Q-learning and Deep Q-network (DQN) to the user device and small base station (SBS) respectively due to different status complexity. The difference between DQN and Q-learning is that instead of using Q-table to store state-action value, DQN uses a neural network to approximate Q-value. However, training and inference of the neural network on small state-action value space consumes more computational resources than the Q-table. Therefore, Q-learning is used on user devices that require more power saving, and DQN is used on SBS for better results. Based on the actor-critic algorithm [31] and multi-agent RL method, Zhao et al. [32] proposed a Neighboring-Aware Edge Caching (NAEC) algorithm, which uses the attention mechanism to selectively learn information from neighboring agents. In general, reinforcement learning-based caching strategy tightly combines model learning and cache strategy, and is more suitable for complex edge caching problem. However, most existing works tend to assume that content popularity is known or predicted by other models before making caching decisions. The former lacks robustness in the time-varying edge caching scenarios. The latter introduces multiple neural network models rather than end-to-end deployment, which introduces additional complexity.

The related references are summarized in Table 1 [33]. And to further improve cache efficiency and reduce cache cost, in this paper, we propose an end-to-end DRL-based caching strategy. The improvement of this paper is that we combine the recurrent neural network and deep reinforcement learning. The recurrent neural network can deal with the temporal characteristic of EC system state and the end-to-end caching strategy simplify the network model. Besides, we take the interests of both users and service providers into account, and the proposed caching strategy can minimize the whole EC cost.

3 System model

In this section, the system architecture of edge content caching is introduced first. Then, the cost of the EC system is presented. The main notations are listed in Table 2.

Table 1 Summary of the related work

Reference	Utilized method	Core considered metrics	Pros	Cons
[20]	Integer programming	The revenue of service provider	Trade-off between service provider and user	Poor comparison
[21]	Game theory	Traffic load	Distributed wireless network caching	Inflexibility
[22]	Prediction-based heuristic algorithm	Average delay and backhaul load	Mobility prediction and collaborative data prefetching	Poor comparison
[23]	Distributed deep learning	Service delay	Distributed training and caching	Poor comparison
[24]	Deep learning	Cache hit rate	Fully consider multiple popularity-related properties	Only hit rate
[26]	Deep learning	Cache hit rate	Joint prediction of user mobility and content popularity	Complexity of two-phase caching
[28]	Federated learning-based DRL	Average utilized	Federated learning	Low caching performance
[29]	GLM prediction and DRL	Cache hit rate and replacement cost	Joint mobility prediction and DRL	Complexity of two-phase caching
[30]	DRL	Energy consumption	Energy consumption	Low caching performance
[32]	Multi-agent DRL	Cache hit rate and average delay	Neighboring-Aware Caching	Complexity

3.1 System architecture

The EC scenario considered in this paper is shown in Fig. 1. There are mainly three different components in this scenario, which are remote cloud data center (DC) in the cloud layer, edge server (ES) in the edge layer, and user equipment (UE). UE can obtain content from a near-end ES. And an ES is a static network facility at the edge of the network, e.g., SBS, RSU. ES communicates with DC and obtains content from DC through the backhaul link. We assume that DC has unlimited computing power and storage resources to store all the content. The biggest difference between ES and traditional cloud DC is the limited compute and storage capacity. Therefore, it is impossible to cache a large amount of content on ES. Besides, the proximity to the UE allows the ES to respond quickly to user requests and improve QoS. This intensifies the competition of ES providers in QoS and cache costs.

UE requests the edge server through a wireless link. The edge server that directly communicates with users is called direct ES. If the requested content has been stored in the direct ES, the direct ES will return the cached content to UE directly. However, if it is not stored in the direct ES, the direct ES first checks whether its neighbor ESs caches the content, if it does, the neighbor ES first sends the content to the direct ES, and then the direct ES returns the content to UE. Otherwise, the direct ES will fetch content from DC, in which case the UE needs to endure the delay of a long backhaul link, resulting in QoS degradation. Assume that each ES has its neighbor ES sets and the distance between the ES and them is much smaller than the distance between the ES and DC. The different content acquisition methods are shown in Fig. 1. The yellow line represents the wireless link between UE and ES. Due to the feature that ES is close to the user, this requires only a small amount of time consumption. The green line indicates that data is transmitted between ESs through high-speed links, with medium time consumption. The blue line indicates that ES obtains content from DC, which has a high time consumption.

3.2 System cost

We divide each time slot t of the ES into two different phases, r_t and c_t, as shown in Fig. 2. In these circumstances, r_t is the phase of collecting UE request information and executing content delivery. Meanwhile, at this phase, ES saves the UE's request information for later agent inferences. ES agent performs inference in the c_t phase to select the caching action. The trained DRL agent infers what should be cached in the next time slot $t+1$ based on the state information observed in r_t, and performs the cache replacement operation. The duration of r_t varies with the number of requests. Besides, each r_t receives the same number of requests N_r and has different lengths of time slot intervals.

Table 2 System notations

Variable	Meaning
d	The content fragment
t	The time slot in which the request arrives at ES
D	The total content collection
K	The total content amounts
N_r	The number of requests in r_t
s_i	The ith edge node
S_i	The set of neighbor ES of i
$D_{s_i,t}$	The content cached by ES s_i in the time slot t
N	The amount of content that an ES can cache
c_{hit}	The network communication cost between UE and direct ES
c_{edge}	The network communication cost between direct ES and neighbor ES
c_{cloud}	The network communication cost between direct ES and DC
c_{COMM}	The total network communication cost
c_{RED}	The cache over storage cost
c_{REPL}	The cache replacement cost
$D_{s_i,t}^{RED}$	The content cached in time slot t but not requested
α	The cache over storage cost factor
D^{repl}	The set of content that needs to be replaced
c_{REPL}^{cloud}	The replacement cost of retrieving data from DC
λ	A binary variable indicates whether content D is cached by neighbor ES set S_i
$c_{REPL}^{S_i}$	The replacement cost of retrieving data from neighbor ES set S_i
C	The total cost of the caching system

On account of the compute and storage capacity limitations of ES, ES can only cache limited contents within the storage capacity. Use r to represent the request of UE and t represents the time slot in which the request arrives at ES. Assume the content can be divided into independent parts like [34], and use d represents one content fragment, meanwhile the total potential content collection is represented by D. Given $D_{s_i,t} = \{d_1, d_2, \ldots, d_n\}$ to represent the set of content cached by the ES s_i in time slot t, and N means the amount of content that an ES can cache. Consequently, the capacity limit of the edge server at time t can be obtained:

$$\sum_{j=0}^{n} d_{j,t} \leq N \tag{1}$$

where $d_{j,t}$ represent the content d_j cached in ES s_i and in time slot t. There are three different cases when a UE requests content d from the ES s_i. First, if the content d requested by

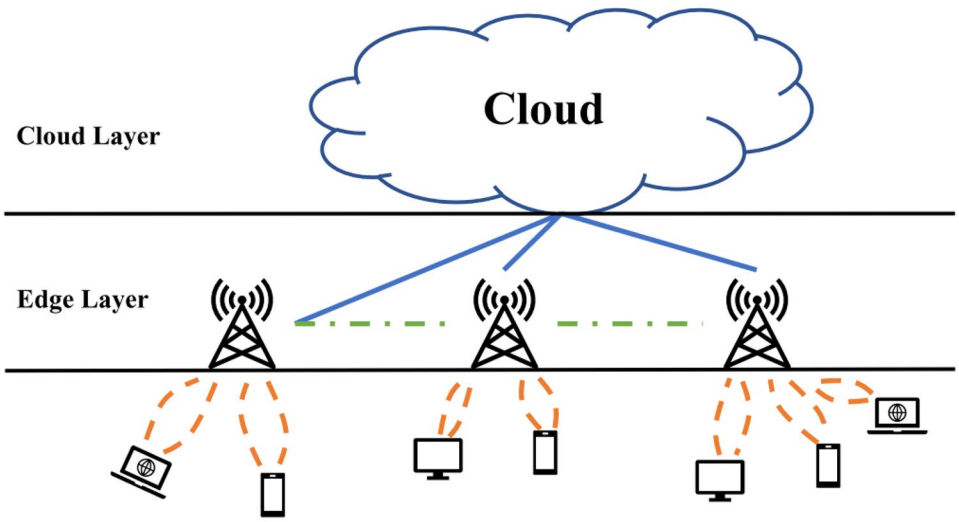

Fig. 1 EC scenario

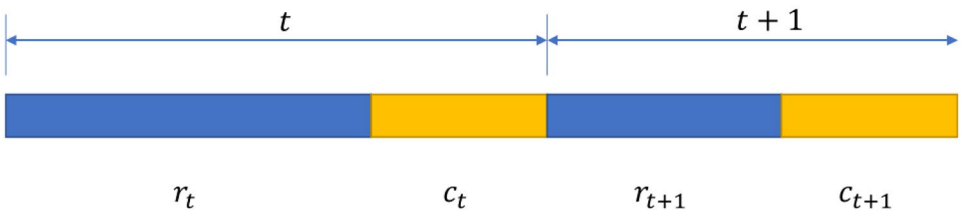

Fig. 2 Time slot division

the UE hits the cache of the direct ES, the network communication cost is defined as c_{hit}. Second, though the content d requested by the UE does not hit the cache of the direct ES, the content is cached by its neighbor ESs S_i, the content will be obtained from its neighbor ES and then return to the user. Hence, the network communication cost is defined as $c_{hit} + c_{edge}$, where c_{edge} is the cost for the direct ES to obtain content from the set of neighbor ES S_i. Finally, if the content d requested by the UE neither hit the cache of the direct ES nor the cache of the neighbor ES set S_i, the cache missed data needs to be obtained from DC. In this circumstance, the network communication cost is defined as $c_{hit} + c_{cloud}$, where c_{cloud} is the cost for the ES to obtain content from DC. Due to the characteristics of edge computing, it is easy to obtain, $c_{hit} \ll c_{edge} \ll c_{cloud}$. To sum up, the total network communication cost can be obtained:

$$C_{COMM} = \begin{cases} c_{hit}, & d \in D_{s_i,t} \\ c_{hit} + c_{edge}, & d \notin D_{s_i,t} \text{ and } d \in D_{S_i,t} \\ c_{hit} + c_{cloud}, & d \notin D_{s_i,t} \text{ and } d \notin D_{S_i,t} \end{cases} \tag{2}$$

In addition to the network communication cost, we also consider the over storage cost of invalid cached content stored at ES. ES agent executes inference to select cache contents $D_{s_i,t+1}$ of the next time slot $t+1$ in c_t, and performs the cache replacement operation. However, due to the limitation of the storage capacity of ES, there will always be cache miss requests in time slot $t+1$. At the same time, there may also be content that has been cached in the ES but has not been requested. Define the set of content that is cached in the ES but not requested by UE in time slot t as, $D^{RED}_{s_i,t} = \{d^{RED}_1, d^{RED}_2, \ldots, d^{RED}_m\}$, $m < n$. Therefore, the contents of the set $D^{RED}_{s_i,t}$ are over-cached. If a content d_j is cached in the ES during time slot $t-1$ and t, and neither is requested, i.e.

$$d_j \in D^{RED}_{s_i,t} \quad and \quad d_j \in D^{RED}_{s_i,t-1} \tag{3}$$

Then this will result in over storage cost in time slot t (this can also occur if multiple identical contents are cached):

$$c_{RED} = \sum_{d_j \in (3)} d_j * \alpha \tag{4}$$

where α is the over storage cost factor, which indicates the degree of cost caused by each over storage content.

Besides, when ES performs the cache replacement operation in c_t in the time slot t, it will cause a cache replacement cost. If the content $d \in D_{s_i,t+1}$ *and* $d \notin D_{s_i,t}$, then the ES needs to transfer new data d from DC or neighbor ES to itself in c_t. Define the set of contents to be replaced as $D^{repl} = \{d^{repl}_1, d^{repl}_2, \ldots, d^{repl}_n\}$. So, the cache replacement cost can be obtained:

$$c_{REPL} = \sum_{d \in D^{repl}} \lambda c^{cloud}_{REPL} + (1-\lambda) c^{S_i}_{REPL} \tag{5}$$

If the content d to be replaced is cached in neighbor ES S_i, then $\lambda = 0$, otherwise, $\lambda = 1$. And due to the characteristics of edge computing, it is assumed that the cost of obtaining data from neighbor ES is always smaller than that from DC, i.e., $c^{S_i}_{REPL} < c^{cloud}_{REPL}$.

In our scenario, obtaining content from the EC system requires comprehensive consideration of network communication cost, over storage cost, and cache replacement cost. Therefore, the total system cost can be obtained by combining the previous discussion:

$$C = c_{COMM} + c_{RED} + c_{REPL} \tag{6}$$

The goal of the EC system is to find an optimal caching strategy π that can minimize the total system cost C.

4 Deep reinforcement learning strategy

In this section, we introduce the novel DDRQN algorithm for intelligent edge caching. At first, we introduce the GRU Layer in the DDRQN agent. Then, we model and formulate the EC problem as MDP. Finally, the DDRQN model is proposed to learn an optimal strategy by interacting with the environment to maximize the system reward, that is, to minimize the total system cost proposed in Sect. 3.

4.1 GRU layer

For the first time, we use GRU based DDQN algorithm to solve the edge content caching problem. We add a GRU layer to the basic DQN agent to learn content popularity in EC scenarios. The GRU algorithm is widely used in time series forecasting, and it can accurately forecast by learning historical information. GRU is a type of RNN, which

is proposed to solve problems such as long-term memory and vanishing gradient in backpropagation. The DDRQN model we proposed uses GRU in the input layer of the agent to replace the fully-connected layer.

Compared with the LSTM algorithm, GRU only requires fewer parameters, but it can get comparable results and is easier to be trained. A GRU cell only contains two gates: reset gate r_t and update gate z_t. The reset gate and the update gate decide how much previous information to be forgotten and how much past information to be passed to the future respectively. In time slot t, the GRU cell takes the current input vector x_t and the hidden state vector h_{t-1} of the previous time slot $t-1$ as input, then output the hidden state h_t of the current time slot t:

$$r_t = \sigma\left(W_r\left[h_{t-1}, x_t\right]\right) \tag{7}$$

$$z_t = \sigma\left(W_z\left[h_{t-1}, x_t\right]\right) \tag{8}$$

$$\widetilde{h}_t = \tanh\left(W\left[r_t \odot h_{t-1}, x_t\right]\right) \tag{9}$$

$$h_t = \left(1 - z_t\right) \odot h_{t-1} + z_t \odot \widetilde{h}_t \tag{10}$$

where r_t, z_t, h_t represent the reset gate, update gate, and hidden state separately; W_r and W_z are the weights of the corresponding unit; tanh is the hyperbolic tangent function and $\odot$ refers to element-wise multiplication.

In our EC scenario, content popularity is unknown. And because of the time-varying characteristic, content popularity in different time slots may vary greatly. So, it is unreasonable to use the content popularity known in the last time slot as the status. Therefore, the system state vector input in our GRU-based DQN agent is expanded from the one-dimensional system state vector S observed in the current time slot to multiple system states $\mathbb{S}$ observed in a period T. That is, the status of the current time slot t is:

$$\mathbb{S}_t = \left[S_{t-T}, S_{t-T+1}, \ldots, S_t\right] \tag{11}$$

The historical state $\mathbb{S}$ is input to the GRU layer, and then the content that needs to be cached in the next time slot is obtained through the fully-connected layer, which is the action $\mathbb{A}_t$. The GRU layer can learn useful information in multiple historical states, especially the time-varying of content popularity, to get better results.

4.2 DDRQN algorithm

4.2.1 MDP

To solve the optimization problem proposed in Sect. 3.2, we first model the EC system as MDP. As represented in Sects. 3.2 and 4.1, the state, action, and reward in MDP are described as follows:

State As mentioned in Sect. 4.1, we define the system state $\mathbb{S}_t$ of the current time slot t as a state matrix composed of the system state S of T time slots. Among them, T depends on the length of the input system state sequence, that is, how many time slots of the system state we need to use in inference. More specifically, the state of each time slot is $S = [P, C], \quad S \in \mathbb{S}$, where p is the request frequency of each content corresponding to the request received in r_t, i.e., the content popularity of t_r, and C represents the current ES cached content set in r_t. Therefore, the system state can be expressed as

$$\mathbb{S}_t = \begin{bmatrix} p^1_{t-T} & \cdots & p^D_{t-T} & c^1_{t-T} & \cdots & c^N_{t-T} \\ p^1_{t-T+1} & \cdots & p^D_{t-T+1} & c^1_{t-T+1} & \cdots & c^N_{t-T+1} \\ \vdots & \cdots & \ddots & \cdots & \cdots & \vdots \\ p^1_{t-1} & \cdots & p^D_{t-1} & c^1_{t-1} & \cdots & c^N_{t-1} \\ p^1_t & \cdots & p^D_t & c^1_t & \cdots & c^N_t \end{bmatrix} \tag{12}$$

The system state matrix composed of multiple time slots will first pass through the GRU Layer mentioned in Sect. 4.1. GRU will learn content popularity and content caching properties by the historical states, and then a fully connected layer will learn to choose a better caching action after GRU.

Action Action is the caching content selected by ES according to the current system state. Assuming that each content has the same size (divide the content into chunks of the same size). ES agent selects the appropriate amount of cache content for the next time slot according to the cache capacity of the ES. According to the system state, the ES agent makes its own choice of action. And based on the reward of environmental feedback, the selected actions are constantly adjusted to minimize long-term costs.

Reward System reward $\mathbb{R}_t$ indicates the pros and cons of the action $\mathbb{A}_t$ that the ES agent makes when facing the system state $\mathbb{S}_t$ in time slot t. In Sect. 3.2, we discussed the system cost C in the EC system. The smaller the system cost, the better the cache effect. To represent the rewards and punishments of the system better, we convert the loss of the communication system into a reward when the cache hits instead of a penalty when the cache is missed, i.e.

$$R_{COMM} = \begin{cases} R_{hit}, & d \in D_{s_i,t} \\ R_e, & d \notin D_{s_i,t}, d \in D_{S_i,t} \\ R_c = 0, & d \notin D_{s_i,t}, d \notin D_{S_i,t} \end{cases} \tag{13}$$

Every time a request hits the cache of direct ES or neighbor ES, the environment will give a positive reward, and $R_{hit} \gg R_e$. Besides, the system over storage cost and cache

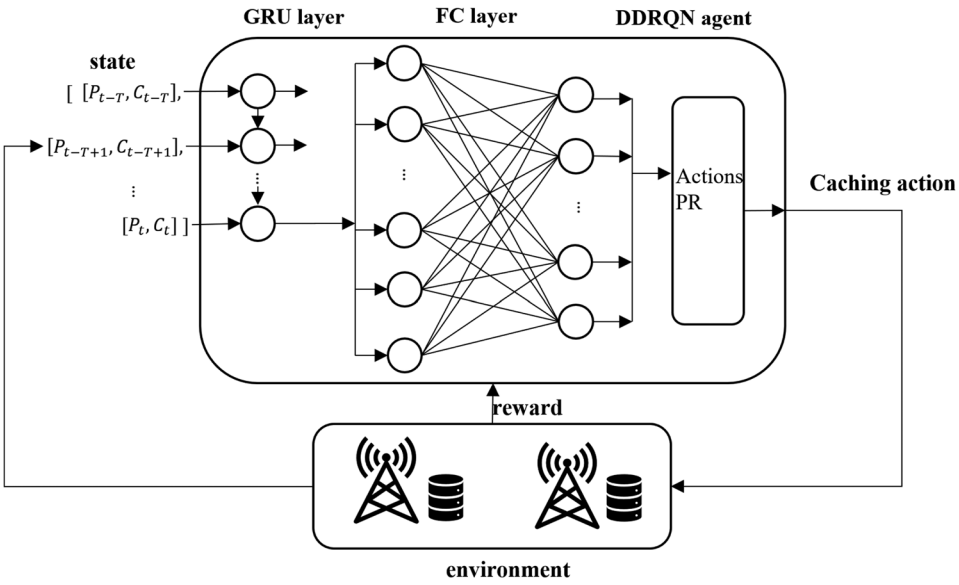

Fig. 3 Interaction between DDRQN agent and EC environment

replacement cost are the same as Sect. 3.2. In this way, the system reward can be obtained:

$$R = R_{COMM} - \left(C_{RED} + C_{REPL}\right) \tag{14}$$

We use the DDRQN algorithm to learn the optimal strategy π. The interaction between the DDRQN agent and the environment can be regarded as MDP, as shown in Fig. 3. Specifically, each ES is a DRL agent, which interacts with the environment (e.g., content popularity, cache state, system state). In each time slot t, the ES agent observes the current system state and selects a caching action according to the EC system state. The environment will do the caching action and generate the next state and reward for the next time slot $t+1$. Then, the agent adjusts its caching policy according to the reward. After the training process, the agent can make cache actions/decisions that maximize long-term rewards while observing the current system state.

4.2.2 DDRQN

In Sect. 4.2, we model the EC system as MDP. Given MDP, the DDRQN agent is shown in Fig. 3. Considering that content popularity plays an indispensable role in the EC scenario, the GRU layer replaces the first fully-connected layer in basic DQN to learn the time-varying content popularity. At the same time the input state of the agent changes to the collection of historical states and observed state, as well as the output of the DDRQN agent is the probability of each caching action. The goal of the DDRQN agent is to find an optimal strategy π. Based on this strategy, the agent can always find the maximum cache action for the expectation of discounted cumulative reward by observing the current state.

Assuming a random strategy π at the beginning of the system,

$$\pi(a|s) = Pr\left(\mathbb{A}_t = a|\mathbb{S}_t = s\right) \tag{15}$$

We can select an action $\mathbb{A}_t$ according to the state of the current time slot $\mathbb{S}_t$. The reinforcement learning system judges the pros and cons of the strategy π according to the expectation of discounted cumulative reward (state-action value function), i.e.

$$Q^{\pi}(s,a) = E\{\sum_{t=0}^{T-1} \gamma^t \mathbb{R}_{t+1} \mid \mathbb{S}_t = s, \mathbb{A}_t = a\} \tag{16}$$

where $T = \infty$ and $\gamma \in [0,1]$ is the discount rate. When γ is close to 0, the agent is more concerned about short-term reward; While it is close to 1, long-term reward become more important. The state-action value function can be obtained by the Bellman equation [35]:

$$Q^{\pi}(s,a) = E\{r\left(s,a,s'\right) + \gamma E\left[Q^{\pi}\left(s',a'\right)\right]\} \tag{17}$$

Then, the optimal state-action value function can be calculated through value iteration:

$$Q^{*}(s,a) = E\{r\left(s,a,s'\right) + \gamma \max_{a'} Q^{*}\left(s',a'\right)\} \tag{18}$$

In this case, the expectation of the discounted cumulative reward of each action can be calculated when observing the

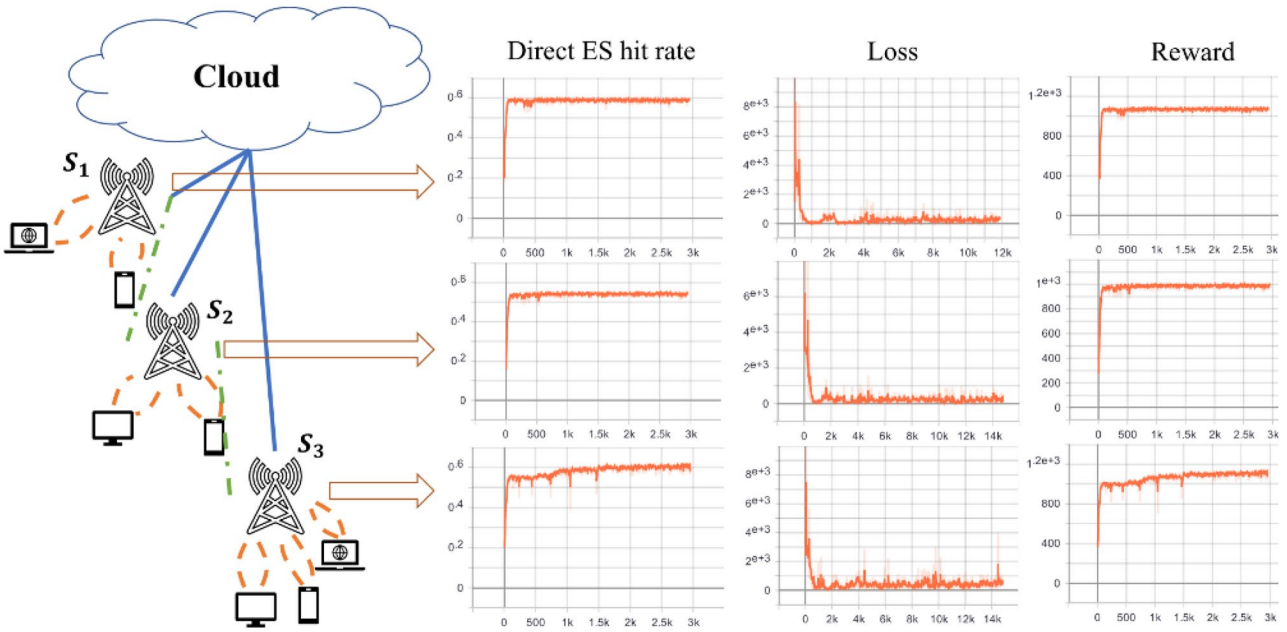

Fig. 4 DDRQN training result

current system state to choose the action with the highest state-action value.

Furthermore, we use the adaptive-ϵ-greedy algorithm in the final action selection, so that the ES agent can balance exploration and exploitation. The ES agent selects the optimal action with a higher probability $1-\epsilon$, and randomly selects an action with a lower probability ϵ. And at the beginning of training, the value of ϵ is larger, so the ES agent will focus more on exploration, mainly selecting caching actions randomly. The value of ϵ decreases by the degree of epsilon decay with the training of the ES agent until its value reaches the threshold of epsilon min. At this time, the trained ES agent mainly selects cache actions through neural network inference, but we still retain a certain probability of random exploration.

Since the state space is too large to store all the Q values with a Q-table, we use the neural network to approximate the Q value [13]. DQN algorithm uses target network and experience replay to solve the unstable and divergent problem in fitting the nonlinear function of the neural network. Moreover, the basic DQN has the problem of overestimating Q value, which may affect the learning of agents. So, the Double-Q-Learning mechanism is proposed to solve overestimating by decoupling the selection and calculation of target Q value [36]. Finally, as mentioned in Sect. 4.1, we have added a GRU Layer to the agent to better learn caching strategies from the state of multiple frames. The proposed DDRQN edge content caching algorithm is given in Algorithm 1 below.

Algorithm 1 DDRQN Algorithm

Randomly initialize Q network weights θ and target Q network weights θ'.
Initialize replay buffer M with the capacity of N_M
Initialize ϵ, γ
for $t = 1$ to T **do**
  Observe initial state $\mathbb{S}_t$ and random probability p
  if $p > \epsilon$ **then**
    Select caching action $\mathbb{A}_t$ according to recurrent neural network
  else
    Select Random caching action $\mathbb{A}_t$
  end if
  Execute caching action $\mathbb{A}_t$ and observe reward $\mathbb{R}_t$ and new state $\mathbb{S}_{t+1}$
  Store transition $(\mathbb{S}_t, \mathbb{A}_t, \mathbb{R}_t, \mathbb{S}_{t+1})$ in M
  Random sample a mini-batch of N transitions j from M
  Using Adam to minimize the MSE loss $\delta_j = (\mathbb{R}_j + \gamma Q'(\mathbb{S}_j, argmax_a Q(\mathbb{S}_j, a)) - Q(\mathbb{S}_{j-1}, \mathbb{A}_{j-1}))^2$
  From time to time copy weights into target network $\theta' \leftarrow \theta$
  Update $\epsilon = \epsilon * decay, \quad if\ \epsilon >= \epsilon_{\min}$
end for

5 Result

In this section, sufficient simulation results are presented to demonstrate the performance of the proposed DDRQN caching strategy.

5.1 Experimental setup

The simulation scenario is shown in Fig. 4. We have a total of 3 edge servers and the total content amount $K=5000$. We set up 3000 epochs to train reinforcement learning agents, each epoch has 100,000 requests, divided into 50 time slots. Therefore, in r_t phase of each time slot, the edge node receives a

total of $N_r = 2000$ requests, and these requests follow the ZipF distribution. The ES needs to give the cached content $D_{S_i,t+1}$ of the size N in the $t+1$ phase in the r_c phase and performs the cache operation. We set the reward $R_{hit} = 1$ for request hitting the direct ES. The reward for not hitting the direct ES but hitting the neighbor ES is $R_e = 0.1$. The reward for missing all caches is $R_c = 0$. And we define the over storage cost factor as $\alpha = 0.5$. As for cache replacement cost, we set the replacement cost of the data obtained from DC is $c_{REPL}^{cloud} = 0.3$, and the replacement cost of obtaining data from neighbor ES is $c_{REPL}^{S_i} = 0.03$.

The neural network structure of the DRL agent is shown in Fig. 3. We design a two-layer network DDRQN algorithm. The input layer is GRU Layer, the hidden size of the GRU Layer is set to 1024 and the time window is set to 24 by default. After the Relu activation function, a fully-connected layer outputs the result of the network, that is, the probability of each content being selected. Therefore, the input dimension of fully-connected layer is equal to 1024 and the output dimension is equal to the ES cache size N, which is set to 300 by default. Besides, the target network update frequent is set to 200, which means the parameters are copied to the fixed network after every 200 steps. We set the discount factor $\gamma = 0.9$, the learning rate of the Adam optimizer is 0.01, and the loss function is Mean Square Error Loss (MSE). At the same time, we set the replay buffer size $N_M = 2000$, the initial value of ϵ to 0.9, the ϵ discount rate to be 0.999, and the minimum value of ϵ to 0.1, which means that the agent is more focused on exploration than exploitation in the early training phase.

5.2 Experimental results

To illustrate the effectiveness and performance of our proposed DDRQN edge content caching algorithm, we conduct experiments under different settings.

Firstly, we present the training proposal of the ES agents using the DDRQN algorithm under the condition of fixed ES cache size and fixed content ZipF distribution parameters. We define the content d follows the ZipF distribution, so the popularity of d can be expressed as: $p(d) = \frac{d^{-a}}{\zeta(a)}$ where $\zeta(a) = \sum_{k=2}^{\infty} 1/k^a$. In this experiment, we set the ZipF parameter $a = 1.2$ and the ES cache size $N = 300$, respectively. We collect the popularity of each content and the content cached by ES in the r_t phase as the system state. Note that all states are known historical data. Due to the huge differences between these data, we perform preprocessing operations on the system state, i.e., normalized the data to between [0,1]. We train three ESs (S_1, S_2, S_3) at the same time. The three ESs can access each other, that is, they are neighbor ESs. If the requested content does not hit the direct ES, the direct ES can retrieve the content from its neighbor ESs. As we can see from Fig. 4, with the increase of training epoch, the cache hit rate of the direct ES agent and the system reward continue to increase and ultimately remain within a certain range, the MSE Loss continues to decline and eventually remains in a lower range near zero. With the growth of training time, under the parameter configuration of this experiment, the DDRQN algorithm can finally obtain a convergence strategy, and the direct cache hit rate of the three ESs finally converges to around 0.6. Eventually, system reward (representing the trade-off between the system's cache hit rate reward and over storage, cache replacement cost) can reach a relatively high level of convergence at average 1062.43, which means that the DDRQN edge cache strategy is in terms of system loss and cache hit rate All have been promoted.

Secondly, we study the effect of different learning rate on DDRQN. The experimental results are shown in Fig. 5. The training performance of the neural network under different learning rate can be seen in the first several training epochs. As shown in Fig. 5, we represent the first 140 epochs to show the training performance. It can be seen that the best training performance of the neural network is achieved when the learning rate is equal to 0.01.

Then, we analyze the performance of the DDRQN algorithm under different ZipF parameter a and fixed ES cache size $N = 300$. As shown in Fig. 6, we evaluate the reward, direct ES cache hit rate, neighbor ES cache hit rate, cache over storage cost rate, and cache replacement cost rate under different a. Note that, cache over storage cost rate, and cache replacement cost rate means their respective total cost divided by the amount of content N the ES can cache. And the following caching strategies are compared.

1. C-DDQN: Wang et al. proposed C-DDQN in [28], where C-DDQN train a centralized agent.

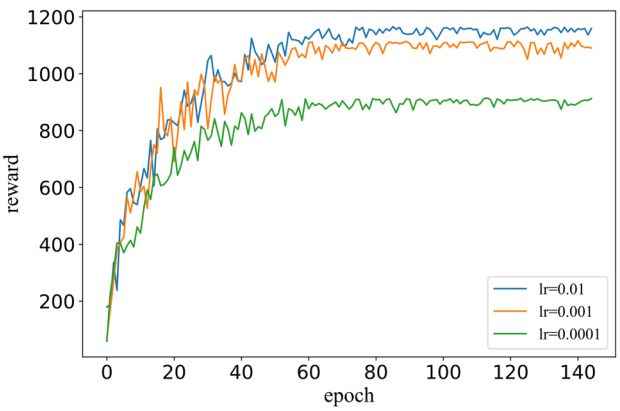

Fig. 5 The effect of different learning rate

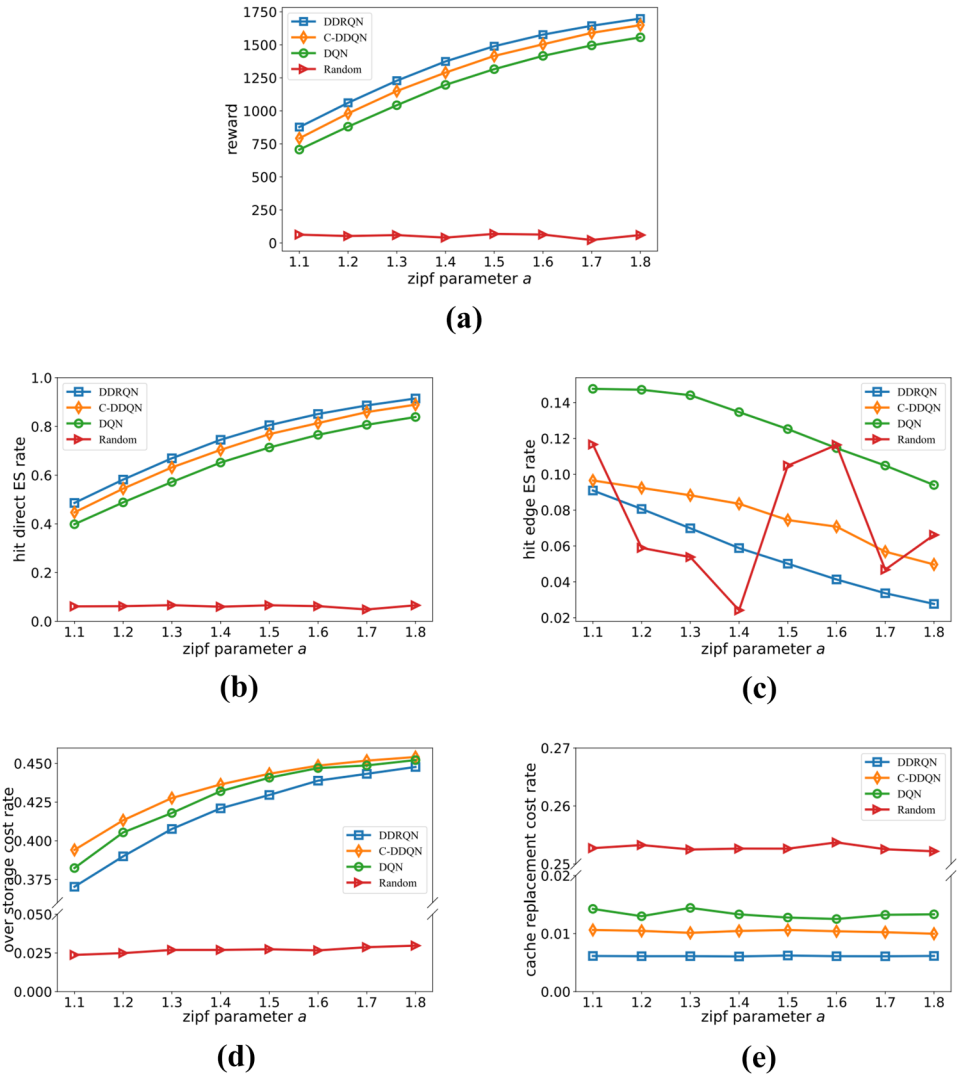

Fig. 6 **a** reward **b** hit direct ES rate **c** hit neighbor ES rate **d** over storage cost rate **e** cache replacement cost rate

2. DQN: DQN is similar to the Q-Learning algorithm proposed in [37] and [38], but due to the large state-action space, DQN uses deep neural network to approximate Q-value rather than Q-table.
3. Random: Random select caching items.

All caching strategies are summarized in Table 3.

From Fig. 6 we can see the caching performance of different algorithms under different content popularity. The larger ZipF parameter a means that the top few popular contents have a greater probability of being requested. Among them, Fig. 6a shows the changes in system rewards as the a increases. With the increase of a, the popular content stored by edge cache servers with the same caching capability has a higher chance of being requested. Therefore, learning-based algorithms can obtain higher system reward, and during the entire experiment, the reward obtained by the proposed DDRQN algorithm are ahead of other algorithms. Figure 6b shows the variation of direct ES cache hit rate with a. When $a = 1.8$, the average cache hit rate of DDRQN reaches 0.91487, which means almost all requests can hit the cache. When $a = 1.1$, the proposed DDRQN caching strategy achieves a system reward of 877 and a direct edge server hit rate of 48.6%, while DQN only achieves a system reward of 707 and a cache hit rate of 39.8%, i.e., the DDRQN caching strategy increases the system reward by 24% and the cache hit rate by 22% compared to the DQN caching strategy.

Table 3 The description of caching strategies

Method	Describe
DDRQN	The caching strategy proposed in this paper
C-DDQN	Centralized-DDQN caching strategy proposed in [28]
DQN	Caching strategy using DQN algorithm
Random	Static caching strategy, random select caching items

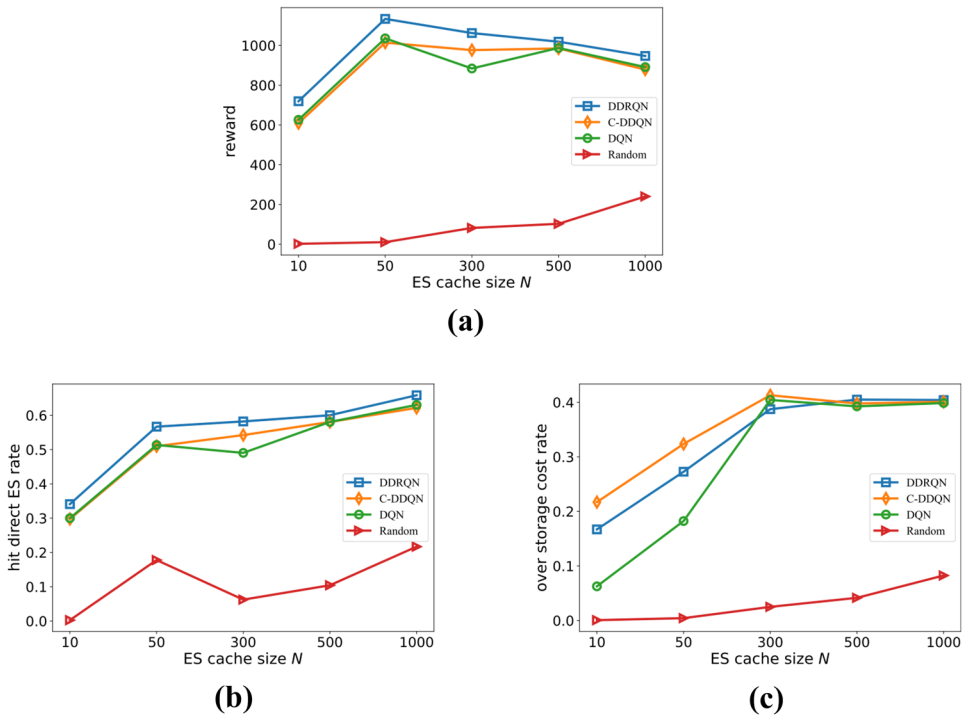

Fig. 7 **a** reward **b** hit direct ES rate **c** hit neighbor ES rate

Due to the increase in the direct ES cache hit rate, only less content needs to be obtained from neighbor ESs or remote DC, so as shown in Fig. 6c, as the a increases, the cache hit rate of neighbor ESs is declining. And since more requests are concentrated on the top few popular contents, the request probability of other sub-popular content cached by the edge server decreases, leading to an incremental over storage cost as shown in Fig. 6d. Finally, as seen in Fig. 6e, the trained agent can always find popular content in the simulation data set, so the cache replacement cost is maintained at the same level and will not change with a. In general, the proposed DDRQN algorithm can get the highest cache hit rate and average system reward under different ZipF parameters a.

Finally, as shown in Fig. 7, we investigate the impact of the ES cache size N on the system performance where the ZipF parameter $a = 1.2$. We can see that the direct ES cache hit rate increases with the improvement of cache capability, and when the ES cache size $N = 1000$, the direct ES cache hit rate of the DDRQN algorithm reaches the highest value of 0.6585. However, the system reward did not increase with the raising of ES cache size. In our experiment, the highest average system reward $\mathbb{R} = 1133.58$ can be obtained when the ES cache size $N = 50$, and then the system reward decreases with the growth of cache size. The reason is that even if the improvement of the ES cache size can lead to an improvement in the direct ES hit rate, the over storage cost of the cache also increases. This shows that under certain content popularity, with the increase of ES cache size, ES caches a lot of unpopular content, which leads to an increase in the storage loss of the system. This is also in line with the limited capacity of edge computing, we only need to use an appropriate amount of space for EC.

6 Conclusion and future work

In this paper, we first study the EC problem with the objective of minimizing system cost. Then the cost of the EC system including network communication cost, over storage cost, and cache replacement cost is investigated. Meanwhile, we model the EC problem as MDP. Finally, the performance of using the DDRQN algorithm in different EC scenarios is presented. And experimental results show the effectiveness of the proposed DDRQN algorithm. In the future work, we will study more complex edge caching problem under real scene and the computational complexity of EC algorithm.

Funding This work is supported by National Science Foundation of China under Grant 61703127, Key R&D Program of Zhejiang Province 2021C01114, and the Graduate Scientific Research Foundation of Hangzhou Dianzi University CXJJ2020087.

Declarations

Conflict of interest All authors in this work declared that they have no conflicts of interest.

References

1. Cisco (2020) Cisco annual internet report (2018–2023) white paper. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html
2. Pham Q-V, Fang F, Ha VN, Piran MJ, Le M, Le LB, Hwang W-J, Ding Z (2020) A survey of multi-access edge computing in 5g and beyond: Fundamentals, technology integration, and state-of-the-art. IEEE Access 8:116974–117017
3. Velte T, Velte A, Elsenpeter R (2009) Cloud computing, a practical approach. McGraw-Hill, Inc.
4. Zhou Z, Chen X, Li E, Zeng L, Luo K, Zhang J (2019) Edge intelligence: Paving the last mile of artificial intelligence with edge computing. Proc IEEE 107(8):1738–1762
5. Xu Y, Gu B, Hu RQ, Li D, Zhang H (2021) Joint computation offloading and radio resource allocation in mec-based wireless-powered backscatter communication networks. IEEE Trans Veh Technol 70(6):6200–6205
6. Zhuang W, Ye Q, Lyu F, Cheng N, Ren J (2019) Sdn/nfv-empowered future iov with enhanced communication, computing, and caching. Proc IEEE 108(2):274–291
7. Xia X, Chen F, He Q, Grundy J, Abdelrazek M, Jin H (2020) Online collaborative data caching in edge computing. IEEE Trans Parallel Distrib Syst 32(2):281–294
8. Nour B, Khelifi H, Moungla H, Hussain R, Guizani N (2020) A distributed cache placement scheme for large-scale information-centric networking. IEEE Network
9. Ale L, Zhang N, Wu H, Chen D, Han T (2019) Online proactive caching in mobile edge computing using bidirectional deep recurrent neural network. IEEE Internet Things J 6(3):5520–5530
10. Wu Z, Lu Z, Zhang W, Wu J, Huang S, Hung PC (2018) A data-driven approach of performance evaluation for cache server groups in content delivery network. J Parallel Distrib Comput 119:162–171
11. Zeng Y, Xie J, Jiang H, Huang G, Yi S, Xiong N, Li J (2019) Smart caching based on user behavior for mobile edge computing. Inf Sci 503:444–468
12. Rathore S, Ryu JH, Sharma PK, Park JH (2019) Deepcachnet: a proactive caching framework based on deep learning in cellular networks. IEEE Network 33(3):130–138
13. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G et al (2015) Human-level control through deep reinforcement learning. Nature 518(7540):529–533
14. Zhong C, Gursoy MC, Velipasalar S (2018) A deep reinforcement learning-based framework for content caching. In: 2018 52nd Annual Conference on Information Sciences and Systems (CISS). IEEE, pp 1–6
15. He X, Wang K, Xu W (2019) Qoe-driven content-centric caching with deep reinforcement learning in edge-enabled iot. IEEE Comput Intell Mag 14(4):12–20
16. Qiao G, Leng S, Maharjan S, Zhang Y, Ansari N (2019) Deep reinforcement learning for cooperative content caching in vehicular edge computing and networks. IEEE Internet Things J 7(1):247–257
17. Hou L, Lei L, Zheng K, Wang X (2018) A q-learning-based proactive caching strategy for non-safety related services in vehicular networks. IEEE Internet Things J 6(3):4512–4520
18. Li S, Li B, Zhao W (2020) Joint optimization of caching and computation in multi-server noma-mec system via reinforcement learning. IEEE Access 8:112762–112771
19. Hausknecht M, Stone P (2015) Deep recurrent q-learning for partially observable mdps. arXiv preprint: arXiv:1507.06527
20. Liu Y, He Q, Zheng D, Xia X, Chen F, Zhang B (2020) Data caching optimization in the edge computing environment. IEEE Trans Serv Comput 1–8
21. Hu Z, Zheng Z, Wang T, Song L, Li X (2016) Game theoretic approaches for wireless proactive caching. IEEE Commun Mag 54(8):37–43
22. Yu G, Wu J (2020) Content caching based on mobility prediction and joint user prefetch in mobile edge networks. Peer-to-Peer Netw Appl 13(5):1839–1852
23. Saputra YM, Hoang DT, Nguyen DN, Dutkiewicz E, Niyato D, Kim DI (2019) Distributed deep learning at the edge: a novel proactive and cooperative caching framework for mobile edge networks. IEEE Wireless Commun Lett 8(4):1220–1223
24. Liang J, Zhu D, Liu H, Ping H, Li T, Zhang H, Geng L, Liu Y (2020) Multi-head attention based popularity prediction caching in social content-centric networking with mobile edge computing. IEEE Commun Lett 25(2):508–512
25. Bernardini C, Silverston T, Festor O (2014) Sonetor: a social network traffic generator. In: 2014 IEEE International Conference on Communications (ICC). IEEE, pp 3734–3739
26. Tang B, Kang L (2021) Eicache: a learning-based intelligent caching strategy in mobile edge computing. Peer-to-Peer Netw Appl 1–16
27. Zhu H, Cao Y, Wang W, Jiang T, Jin S (2018) Deep reinforcement learning for mobile edge caching: Review, new features, and open issues. IEEE Network 32(6):50–57
28. Wang X, Han Y, Wang C, Zhao Q, Chen X, Chen M (2019) In-edge AI: Intelligentizing mobile edge computing, caching and communication by federated learning. IEEE Network 33(5):156–165
29. Zhang N, Zheng K, Tao M (2018) Using grouped linear prediction and accelerated reinforcement learning for online content caching. In: 2018 IEEE International Conference on Communications Workshops (ICC Workshops). IEEE, pp 1–6
30. Tang J, Tang H, Zhang X, Cumanan K, Chen G, Wong K-K, Chambers JA (2019) Energy minimization in d2d-assisted cache-enabled internet of things: a deep reinforcement learning approach. IEEE Trans Industr Inf 16(8):5412–5423
31. Ye Z, Zhang D, Wu Z-G, Yan H (2021) A3c-based intelligent event-triggering control of networked nonlinear unmanned marine vehicles subject to hybrid attacks. IEEE Trans Intell Transport Syst
32. Zhao Y, Li R, Wang C, Wang X, Leung VC (2021) Neighboring-aware caching in heterogeneous edge networks by actor-attention-critic learning. In: ICC 2021-IEEE International Conference on Communications. IEEE, pp 1–6
33. Xu Y, Xie H, Li D, Hu RQ (2022) Energy-efficient beamforming for heterogeneous industrial iot networks with phase and distortion noises. IEEE Trans Ind Inform
34. Ghemawat S, Gobioff H, Leung S-T (2003) The google file system. In: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, pp 29–43
35. Watkins CJ, Dayan P (1992) Q-learning. Mach Learn 8(3–4):279–292
36. Van Hasselt H, Guez A, Silver D (2015) Deep reinforcement learning with double q-learning. arXiv preprint: arXiv:1509.06461
37. Zhang D, Ye Z, Chen P, Wang Q-G (2020) Intelligent event-based output feedback control with q-learning for unmanned marine vehicle systems. Control Eng Pract 105:104616
38. Zhang D, Ye Z, Feng G, Li H (2021) Intelligent event-based fuzzy dynamic positioning control of nonlinear unmanned marine vehicles under dos attack. IEEE Trans Cybern

Haitao Xu, associate professor. His research interests are in the areas of intelligent transportation system, machine learning and data mining. As a member, he won the second prize of Zhejiang Science and Technology Award and the first prize of Zhejiang Higher scientific research achievement awards.

Yuejun Sun is a postgraduate student at the School of Software Engineering, Hangzhou Dianzi University. His research interests are edge computing and edge intelligence.