Intelligent Cooperative Caching at Mobile Edge based on Offline Deep Reinforcement Learning

ZHE WANG, JIA HU, and GEYONG MIN, University of Exeter, United Kingdom
ZHIWEI ZHAO, University of Electronic Science and Technology of China, China

Cooperative edge caching enables edge servers to jointly utilize their cache to store popular contents, thus drastically reducing the latency of content acquisition. One fundamental problem of cooperative caching is how to coordinate the cache replacement decisions at edge servers to meet users' dynamic requirements and avoid caching redundant contents. Online deep reinforcement learning (DRL) is a promising way to solve this problem by learning a cooperative cache replacement policy using continuous interactions (trial and error) with the environment. However, the sampling process of the interactions is usually expensive and time-consuming, thus hindering the practical deployment of online DRL-based methods. To bridge this gap, we propose a novel Delay-awarE Cooperative cache replacement method based on Offline deep Reinforcement learning (DECOR), which can exploit the existing data at the mobile edge to train an effective policy while avoiding expensive data sampling in the environment. A specific convolutional neural network is also developed to improve the training efficiency and cache performance. Experimental results show that DECOR can learn a superior offline policy from a static dataset compared to an advanced online DRL-based method. Moreover, the learned offline policy outperforms the behavior policy used to collect the dataset by up to 35.9%.

CCS Concepts: • **Networks → Network services**; • **Computing methodologies → Reinforcement learning**.

Additional Key Words and Phrases: Offline deep reinforcement learning, cache replacement, convolutional neural network, edge computing, smart city

ACM Reference Format:
Zhe Wang, Jia Hu, Geyong Min, and Zhiwei Zhao. 2023. Intelligent Cooperative Caching at Mobile Edge based on Offline Deep Reinforcement Learning. *J. ACM* 37, 4, Article 111 (August 2023), 24 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

As one of the prominent characteristics of the smart city, the ever-increasing amount of mobile data traffic is constantly being generated by various mobile devices and smart applications from different sectors like industry [43], healthcare [47], transportation [20], and social network [44]. Ericsson predicts that mobile data traffic is expected to grow around 4.2x from 2021 to 2027 [5]. Such a vast amount of mobile data traffic poses grand challenges on the communication network of

This work was supported in part by UKRI Grant No. EP/X038866/1 and Horizon EU Grant No. 101086159. For the purpose of open access, the author has applied a Creative Commons Attribution (CC BY) licence to any Author Accepted Manuscript version arising.
Author's addresses: Z. Wang, J. Hu (corresponding author), and G. Min (corresponding author), Department of Computer Science, University of Exeter, United Kingdom; emails: {zw329, j.hu, g.min}@exeter.ac.uk; Z. Zhao, School of Computer Science and Engeeneering, University of Electronic Science and Technology of China, China; email: zzw@uestc.edu.cn.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2023 Association for Computing Machinery.
0004-5411/2023/8-ART111 $15.00
https://doi.org/XXXXXXX.XXXXXXX

the smart city, causing a series of problems such as infrastructure overload and network congestion, which inevitably reduces Quality-of-Experience (QoE) to be perceived by users.

Mobile edge caching is one of the core technologies of the smart city to solve the above problem by storing contents in cache-equipped edge servers closer to users [24]. Therefore, users can directly obtain contents from the nearby edge servers instead of the remote cloud, reducing the transmission delay and improving the QoE. Due to the limited storage of the edge servers, only popular contents are cached on the edge servers to exploit the potential of the mobile edge caching as much as possible. However, the content popularity is usually unknown in advance and varies with user preferences. Thus the edge servers need to determine how to continuously replace the cached contents with new popular ones to meet users' dynamic requirements. Although the cache replacement problem has been exhaustively studied [36], most of them focus on a single edge server scenario, requiring each edge server in the smart city executes an independent cache replacement policy. However, such approaches would result in an insufficient utilization of the limited storage since multiple edge servers may redundantly store the same popular contents [32]. A practical solution is to enable the edge servers to share the cached contents with their nearby servers for cooperative edge caching. In this case, an edge server may tend to cache contents other than the popular contents already cached in a nearby server to meet more requirements [3]. Compared with the single edge cache replacement problem, the cooperative cache replacement problem should also consider the coordination between servers to avoid caching too many redundant contents, and it is proven to be NP-hard [39].

Reinforcement learning (RL), developed for solving decision-making problems, is naturally a promising approach for solving the cooperative cache replacement problem. By introducing deep neural networks (DNNs) with powerful representation ability [42], deep RL (DRL) can effectively handle the complex communication network state of the smart city. Recently, online DRL has been proven to automatically learn a cooperative cache replacement policy by continuously interacting with the environment [27, 30, 35]. However, these works emphasize the superiority of online DRL but ignore the potential risks brought by the exploration nature of online DRL. To find an excellent cooperative cache replacement policy, the online DRL agent would execute different replacement decisions to explore better decisions. During this process, it also inevitably explores poor decisions, thus degrading the cache performance. In addition, the policy requires a large number of interactions to be well-trained, and the interaction collection is expensive and time-consuming.

To address the above two challenges, we propose a novel Delay-awarE Cooperative cache replacement method based on Offline DRL (DECOR). DECOR decouples the interaction phase from the policy training, so no matter how bad the policy becomes during training, it does not affect the cache performance in the actual environment. In addition, the policy can be directly trained with historical log data that already existed in the network without interacting with the environment.

- We develop a powerful and effective scheme named DECOR for cooperative cache replacement scheme, making use of advanced offline DRL to improve cache performance and enhance user experience. Compared to the advanced online DRL-based method, DECOR can successfully learn a superior policy from a static dataset without interacting with the environment.
- We formulate the problem of real-time cooperative cache replacement without prior knowledge of content popularity as a delay-minimization problem, where the effect of the delayed hits mechanism in reducing transmission delay is explicitly discussed. To handle the uncertainties and dynamic nature of the environment, we model the optimization problem as a Markov Decision Process (MDP) to support solving by the offline DRL.
- We design a multi-head Convolutional Neural Network (CNN) to achieve parallel processing of content popularity for different servers while preventing information interference among

them. Each head in the network is dedicated to extracting temporary features of content popularity from the data associated with a specific edge server.
- We undertake extensive experiments to thoroughly validate the efficiency of DECOR. The user requests and preferences are simulated using a real-world dataset called Movielens. The experimental results reveal that the offline policy learned by DECOR surpasses both the default behavior policy and the policy learned by an advanced online DRL-based method in interactions with a static dataset.

The rest of this paper is organized as follows: The related work on the cooperative cache replacement problem is reviewed in Section 2. Section 3 briefly introduces the background of RL and delayed hits. Section 4 discusses the system model including three parts: network, fetching delay, and delivery delay models, and then presents the problem formulation of cooperative edge caching with delayed hits. The details of DECOR are shown in Section 5. Section 6 describes the experimental setting and gives detailed analysis of the experimental results. Finally, the conclusion of this paper is presented in Section 7.

2 RELATED WORK

The cooperative cache replacement problem has attracted many researchers to fully utilize the insufficient storage of edge servers [34]. Since the problem is NP-hard, heuristic-based methods are feasible candidates for solving it in low time complexity, thus meeting users' dynamic requirements promptly. Wang *et al.* [28] developed a heuristic cooperative cache replacement scheme for zone-based cooperative content caching and delivering in large radio access networks. The scheme divides the edge server storage into two parts to separately cache locally popular contents and globally popular contents. Zhang *et al.* [38] proposed a greedy cooperative caching algorithm to proactive replace the cached layered video files during off-peak hours to reduce the total transmission delay. However, these works have an impractical assumption that the content popularity is known in advance. Some researchers tried extracting the content popularity from historical information to address this issue. The solution developed in [40] predicts user preferences based on the users' request history and social information, which is further used to optimize the cache replacement policy into a low-complexity heuristic algorithm. Although the performance of heuristic-based methods is usually good enough, their design relies heavily on an expert understanding of the environment and requires too much time and human resources.

DRL has already yielded remarkable results in solving cooperative cache replacement problems in recent years [7]. It can automatically learn the cooperative cache replacement policy by interacting with the environment without knowing the content popularity. Xu *et al.* [35] proposed a deep deterministic policy gradient-based algorithm to decide the cache decisions and plan users' mobility trajectories inside a cooperative caching domain in a global view and developed a cross-domain content delivery method. Instead of the content popularity-based policy, Song *et al.* [25] proposed a QoE-driven cache replacement method based on DRL in the cooperative edge caching-aided vehicle networks. Unlike the above centralized DRL, a multi-agent DRL is used in [27] to enable each edge to learn its optimal local policy while cooperating with other edges in a decentralized manner. Wang *et al.* [30] used a deep Q-learning (DQN) network to jointly decide the cache node selection and content replacement for a device-to-device assisted collaborative edge caching scenario. The approach incorporates DQN into a federated learning framework to improve training efficiency. The DRL methods used in the above work operate in an online fashion, requiring continuous interaction with the environment to sample a large amount of data for training an ideal agent. Unfortunately, the sampling process is typically costly and time-consuming. Moreover, the trial-and-error nature of

DRL may lead to performing poor cache replacement decisions during the online training process, thereby deteriorating the performance of the edge caching system.

To cope with the limitations of online DRL, offline DRL has been proposed and has demonstrated excellent and reliable performance in different fields [22]. A decentralized approach based on the offline DRL algorithm has been proposed in [10] to address the task of cooperative spectrum sensing in a cognitive radio network. In this approach, each secondary user utilizes a local Conservative Q-Learning (CQL) model to determine the presence of licensed users based on their local sensing, while a fusion center employs another CQL model to make a global decision by aggregating the results from all local CQL models. Zhan *et al.* [37] developed a model-based offline RL framework for optimizing the combustion efficiency of thermal power generating units. The framework initially trains a simulator offline using real datasets and subsequently utilizes a combination of real and simulated datasets to train the RL agent offline. Xiao *et al.* [33] formulated the interactive recommendation as a probabilistic inference problem. To solve this problem, the authors incorporated several regularization techniques into an online DRL algorithm to train an RL agent offline. While offline DRL has been successfully applied in various real-world scenarios, there remains a notable gap in its application to the edge caching replacement problem.

3 BACKGROUND

‘

3.1 Reinforcement Learning

RL has recently shown extraordinary potential in many domains, such as wireless charging [17], video games [18], task scheduling [45], and computation offloading [23]. The decision-making problem solved by RL is modeled as an MDP that is denoted as a 5-tuple $< S, A, T, R, \gamma >$, where S, A, $T : S \times A \longrightarrow S$, $R : S \times A \longrightarrow \mathbb{R}$ are the state space, action space, state transition probability matrix, and reward function, respectively. $\gamma \in (0, 1)$ is a discount factor. At each time step t, the RL agent receives a state s_t from the environment and follows a policy $\pi(a_t|s_t)$, which is a probability of taking action a_t when receiving the state s_t, to produce an action a_t to the environment. The environment executes a_t and transits to the next state a_t by following the state transition probability matrix $T(s_{t+1}|s_t, a_t)$, then generates an immediate reward $r_t(s_t, a_t)$ simultaneously. The environment returns the new state s_{t+1} and reward $r_t(s_t, a_t)$ to the RL agent. The RL agent repeats the above interactions until the decision-making process ends and samples a trajectory $\tau^\pi = (s_0, a_0, r_0, s_1, a_1, r_1)$. We can obtain the accumulated reward when starting at the time step t as follows:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}. \tag{1}$$

The true action value $q^\pi(s_t, a_t) = \mathbb{E}_\pi[G_t|s_t, a_t]$ of the policy π, also regarded as the Q-value, is the expected accumulated reward when following the policy after executing action a_t for a given state s_t. It evaluates how good a state-action pair is and can guide how to select actions to achieve the highest accumulated reward. However, due to a large number of trajectories, it is challenging to compute $q^\pi(s_t, a_t)$ directly with them. We use a Q-function $Q^\pi(s_t, a_t)$ to approximate $q^\pi(s_t, a_t)$ and calculate it through Bellman Operator B_π:

$$B_\pi Q^\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}}[r_t + \gamma Q^\pi(s_{t+1}, \pi(a_{t+1}|s_{t+1}))]. \tag{2}$$

$Q^\pi(s_t, a_t)$ eventually converges to the true Q-value $q^\pi(s_t, a_t)$ by continually updating using the operator. An optimal function can be determined by $Q^*(s_t, a_t) = \max_\pi Q^\pi(s_t, a_t)$, and its corresponding optimal policy, which RL aims to find, can be obtained by $\pi^*(a_t|s_t) = \arg\max_{a_t} Q^*(s_t, a_t)$.

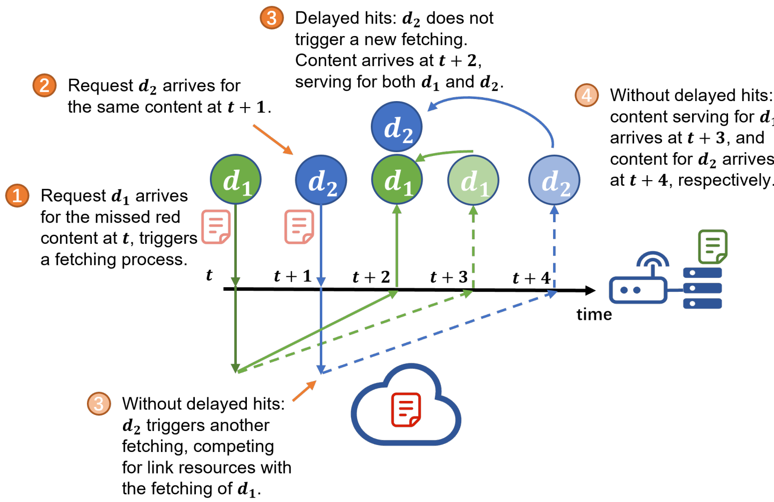


Fig. 1. Example of the edge caching with/without delayed hits. Both have the same steps 1-2. Transparent step 3 uses the delayed hits mechanism, but opaque steps 3-4 do not use it.

3.2 Delayed Hits

The workflow of edge caching is that: when users send requests to an edge server for some contents, if the contents are cached in the server (regarded as cache hits), the server can return them to users immediately. If the contents do not exist in the server (regarded as cache misses), the fetching processes for the missed contents are triggered to retrieve them from the cloud. When the missed contents arrive at the server from the cloud, the server returns them to the users.

In traditional edge caching-aided network models, the fetching processes triggered by different requests are independent, leading to intense competition for the link resources between the cloud and edge servers, thus significantly increasing the transmission delay of requests. The problem is even more severe in a massive explosion of requests like the smart city. The delayed hits mechanism is a simple but efficient method to solve the problem: when a request for a missed content triggers a fetching process, all subsequential requests for the same missed content share the fetching process without triggering new ones before the missed content has been retrieved from the cloud [1]. A simple example of the differences between edge caching with/without delayed hits is illustrated in Fig. 1. Suppose that the content size of the red content is RC unit, and the bandwidth of the link between the cloud and edge server is $\frac{1}{2RC}$ unit per second. At any given time, each user can only have one request being served. At time t, a request d_1 from one user arrives at the edge server for the red content, triggering a fetching process. The process is expected to be completed at time $t+2$. At time $t+1$, before the fetching process triggered by d_1 is completed, a second request d_2 from another user arrives at the server, triggering a new fetching process for the same content if the delayed hits mechanism is not used. The new fetching process competes for the link resource with the previous one, causing the fetching delay of both to be extended. As a result, the fetching delay of d_1 and d_2 is 3. When the delayed hits mechanism is introduced, d_2 chooses to wait for the completion of the fetching process triggered by d_1 instead of arising a new one. Then, the red content arrives at $t+2$ to serve both d_1 and d_2. In this case, the fetching delay of d_1 and d_2 is reduced to 2 and 1, respectively.

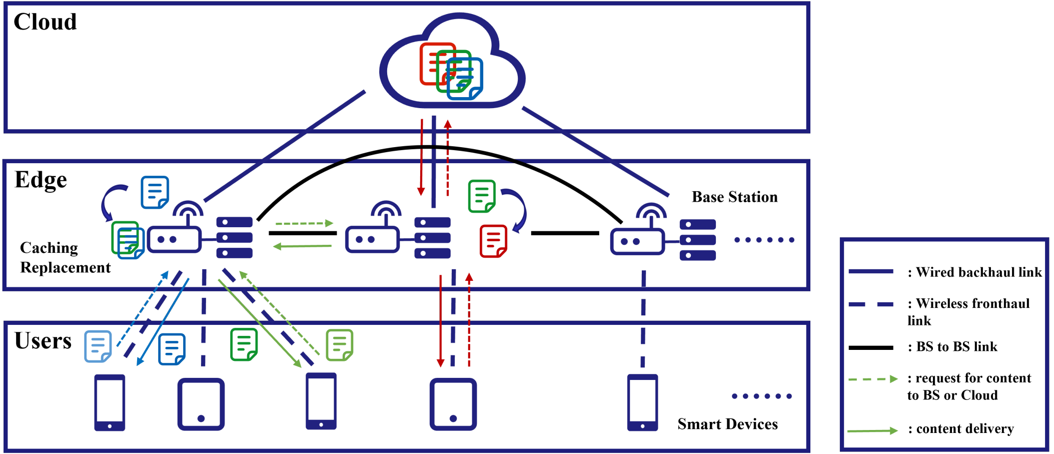


Fig. 2. Architecture of the cooperative edge caching-aided mobile edge computing network. The dashed arrows represent the process of content request for light-colored contents, and the solid arrows represent the process of content delivery for dark-colored contents. The colored arrows represent the request/delivery processes of content corresponding to their respective colors.

4 SYSTEM MODEL

In this section, we first present the network model of the cooperative edge caching-aided mobile edge computing (MEC) network. Then, the fetching and delivery delay models are discussed in detail. Finally, we formulate the cooperative cache replacement problem as an optimization problem based on the above three models.

4.1 Network Model

As shown in Fig. 2, we consider a fundamental cooperative edge caching-aided MEC network which consists of a remote cloud with unlimited cache capacity, K base stations (BSs) equipped with cache-aided edge servers, and multiple users, where the BSs are connected with the cloud via wired backhaul links, the BSs are connected with others via wired bi-directional links, and the users are communicated with the BSs via wireless fronthaul links. The topology of BSs is defined as $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V} = \{v_1, v_2, ..., v_k, ...v_K\}$ is the set of BSs with size K, and $\mathcal{E} = \{e_{1,1}, e_{1,2}, ..., e_{K,K-1}, e_{K,K}\}$ is the set of connections between BSs with size $K \times K$. If v_m is connected to v_n, $e_{m,n} = e_{n,m} = 1$ and its corresponding link bandwidth $b_{m,n} = b_{n,m} > 0$; Otherwise $e_{m,n} = e_{n,m} = 0$ and $b_{m,n} = b_{n,m} = 0$. It should be noticed that $e_{m,m} = 0$. We consider v_m and v_n are neighboring if $e_{m,n} = e_{n,m} = 1$. Each BS covers a specific area of users to serve them, and the service coverage of all BSs is non-overlapping. The bandwidth size of the backhaul links is the same as $b_{0,k} = b_{k,0} = BD$, where 0 is the index of the remote cloud.

We assume that there is a content library containing a series of contents with different indexes $C = \{1, 2, ..., C\}$ in the remote cloud, and there are N users requesting the contents in the library. All contents have the same size as 1 unit [11, 46], and we can divide contents of different sizes into multiple small slices with the same size to achieve this assumption. It is assumed that the limited storage space of all BSs has the same size as L unit, where $L << C$ indicates that only a small portion of the contents can be cached on the BSs.

The service process of the cooperative edge caching-aided MEC network will last for a long term. During this term, the contents in the library are constant. To detect changes in MEC network status in real-time, we equally split the service term into multiple small timeslots, denoted as $\mathcal{T} = \{1, 2, ..., t, ..., T\}$, and record the network status at the beginning and the end of each timeslot. This time division method can also be extended to a service process that does not have an explicit termination time but does have a terminal state. In this case, T represents the timeslot when the service process reaches the terminal state. At the beginning of the timeslot t, the BS v_k receives some new requests $\mathcal{A}^{t,k} = \{a_{1,1}^{t,k}, a_{1,2}^{t,k}, ..., a_{i,j}^{t,k}, ..., a_{N,C-1}^{t,k}, a_{N,C}^{t,k}\}$ from users without knowing the content popularity, where $a_{i,j}^{t,k} \in \{0,1\}$. $a_{i,j}^{t,k} = 1$ means that a new request demanding the content j from the user i to the BS v_k is raised in the timeslot t; Otherwise, $a_{i,j}^{t,k} = 0$. Apart from these newly arrived requests, v_k has some old requests that still need to be processed. We define an entire request set of v_k at the beginning of the timeslot t to contain all old and new requests as $\mathcal{D}^{t,k} = \{d_{1,1}^{t,k}, d_{1,2}^{t,k}, ..., d_{i,j}^{t,k}, ..., d_{N,C-1}^{t,k}, d_{N,C}^{t,k}\}$, where $d_{i,j}^{t,k} \in \{0,1\}$. $d_{i,j}^{t,k} = 1$ means that a request demanding the content j from the user i exists on the v_k at the beginning of the timeslot t. Similarly, the request set of v_k at the end of the timeslot t can be defined as $\mathcal{D}'^{t,k} = \{d'^{t,k}_{1,1}, d'^{t,k}_{1,2}, ..., d'^{t,k}_{N,C}\}$. In each timeslot, only one request per user can be served by the MEC network:

$$\sum_k \sum_j d_{i,j}^{t,k} \leq 1, \sum_k \sum_j d'^{t,k}_{i,j} \leq 1, \forall t, i; \tag{3}$$

Besides, users can raise new requests only if their existing requests are successfully processed:

$$\sum_k \sum_j (d'^{t,k}_{i,j} + a_{i,j}^{t+1,k}) \leq 1, \forall t, i; \tag{4}$$

According to the above constraints, we can deduce that:

$$d_{i,j}^{t+1,k} = d'^{t,k}_{i,j} + a_{i,j}^{t+1,k}. \tag{5}$$

In the timeslot t, the set of contents cached on v_k is denoted as $U^{t,k}$, which is also regarded as the cache strategy of v_k. If the content j is cached on v_k during the timeslot t, then $j \in U^{t,k}$; Otherwise, content $j \notin U^{t,k}$ and it is missed on v_k. According to the workflow of edge caching, we can find that the transmission delay of requests is divided into three parts: request sending delay, content fetching delay, and content delivery delay. Since the size of requests is much smaller than the size of contents, compared with the content fetching delay and content delivery delay, we can ignore the request sending delay.

4.2 Fetching Delay Model

In this model, we explicitly consider the impact of delayed hits. When multiple fetching processes are triggered for the same missed content, only one fetching process is executed. The current requests and the subsequential arrived requests for the missed content share this fetching process before the process ends. For each BS, each content corresponds to one fetching process. Considering that BS can fetch missed contents from the cloud or the neighboring BSs, we use $\mathcal{X}_j^{t,k} = \{x_j^{t,(0,k)}, x_j^{t,(1,k)}, ..., x_j^{t,(m,k)}, ...x_j^{t,(K,k)}\}, m \neq k$ to indicate where v_k fetches the missed contents from. $x_j^{t,(m,k)} \in \{0,1\}$ where $x_j^{t,(m,k)} = 1$ means that v_k fetches the missed content j from server v_m in the timeslot t. When $m = 0$, v_k fetches the missed content from the cloud. If v_k is not connected to v_m, it cannot fetch contents from v_m, so we have the following:

$$x_j^{t,(m,k)} \leq e_{m,k}, \forall t, j; \tag{6}$$

Table 1. Summary of Main Notations

Notation	Description
$\mathbb{I}(\cdot)$	The value is 1 if the condition within parentheses is true; Otherwise, the value is 0.
$\mathcal{C}$	The content library in the remote cloud.
C	The total number of content types.
K	The number of BSs.
L	The cache storage size of each BS.
$A^{t,k}$	The set of requests arriving at the BS v_k in the timeslot t.
$a_{i,j}^{t,k}$	The value is 1 if a new request from the user i for the content j arrives at v_k at the beginning of the timeslot t; Otherwise, the value is 0.
$D^{t,k}$	The set of requests exists on v_k at the beginning of the timeslot t.
$d_{i,j}^{t,k}$	The value is 1 if the request from the user i for the content j exists on v_k at the beginning of the timeslot t; Otherwise, the value is 0.
$D'^{t,k}$	The set of requests exists on v_k at the end of the timeslot t.
$d'^{t,k}_{i,j}$	The value is 1 if the request from the user i for the content j exists on v_k at the end of the timeslot t; Otherwise, the value is 0.
$X_j^{t,k}$	Determine the source node of the fetching process for the missed content j.
$x_j^{t,(m,k)}$	The value is 1 if v_k fetches the missed content j from v_m in the timeslot t; Otherwise, the value is 0.
$U^{t,k}$	The cache strategy of v_k in the timeslot t.
$O'^{t,k}$	The set of contents fetched from v_k's neighboring BSs or the remote cloud at the end of the timeslot t.
$F_j^{t,k}$	The remaining size of the content j to be fetched from v_k at the beginning of the timeslot t.
$F'^{t,k}_j$	The remaining size of the content j to be fetched from v_k at the end of the timeslot t.
$r_m^{t,k}$	The average fetching rate from v_m to v_k in the timeslot t.
π	The decision-making policy for replacing cached contents of all BSs with their newly arrived contents.
$Flag_{i,j}^{t,k}$	The value is 1 if v_k delivers the content j to the user i in the timeslot t; Otherwise, the value is 0.
$V_{i,j}^{t,k}$	The remaining size of the content j to be delivered from v_k to the user i in the timeslot t.
$r_i^{t,k}$	The delivery rate from v_k to the user i in the timeslot t.

Since there is at most one fetching process per BS for the same content in each timeslot, we also have:

$$\sum_{m}^{m\neq k} x_j^{t,(m,k)} \leq 1, \forall t, j, k; \tag{7}$$

The delayed hits mechanism can effectively reduce the frequency of fetching the same content but cannot avoid the fetching of different contents. When multiple fetching processes for different contents coexist, they inevitably compete for bandwidth resources. This problem becomes more complex when cooperation is required among multiple BSs. Furthermore, over time, some ongoing

fetching processes will be resolved while some new processes are triggered, resulting in dynamic changes in the fetching rate for each process. To solve the above problems, our model needs to perceive the state of each fetching process in real-time. The fetching process of content j at the beginning of the timeslot t on the BS v_k is recorded by using the remaining size of the content j that still needs to be fetched, denoted as $F_j^{t,k}$. Meanwhile, we use $F_j^{\prime t,k}$ to record the remaining size of the content j at the end of the timeslot t. $F_j^{t,k}$ can be determined according to two different cases: in timeslot t, (1) there are new requests for a missed content j to trigger a new fetching process on the BS v_k; (2) there is no new fetching process triggered on v_k. For case (1), when there does not already exist a fetching process for the content j, the content j is not cached on the BS v_k, and new requests for the missed content j arrive at v_k, a new fetching process is triggered, and the remaining size $F_j^{t,k}$ becomes to 1:

$$F_j^{t,k}(new) = \mathbb{I}(F_j^{\prime t-1,k} == 0) \times \mathbb{I}(\sum_i a_{i,j}^{t,k} > 0) \times \mathbb{I}(j \notin U^{t,k}), \tag{8}$$

where $\mathbb{I}(\cdot) = 1$ if and only if the condition within parentheses is true; Otherwise, it is 0.

When a new fetching process for the missed content j is triggered, we must determine where to fetch it. Here, we provide a simple source node selection method. Since all fetching processes equally share the the BS-to-BS links between the BSs, we can easily calculate the average fetching rate of all links from v_k's neighboring BSs to v_k in the timeslot $t-1$ by:

$$r_m^{t-1,k} = \frac{b_{m,k}}{\max\{\sum_j x_j^{t-1,(m,k)}, 1\}}, \tag{9}$$

where the max operator ensures that if there is no fetching process from the neighboring BS to v_k, the average fetching rate equals the link bandwidth per timeslot. Then we choose the neighboring BS that owns the highest average fetching rate in the timeslot $t-1$ and has cached the missed content j in the timeslot t as the source node:

$$x_j^{t,(m,k)} = \begin{cases} e_{m,k} \times \mathbb{I}(j \in U^{t,m}) \times \mathbb{I}(m == \arg\max_n \{r_1^{t-1,k}, ..., r_n^{t-1,k}, ..., r_K^{t-1,k}\}), & if \;\; F_j^{t,k}(new) == 1; \\ x_j^{t-1,(m,k)}, & if \;\; F_j^{\prime t-1,k} > 0; \\ 0, & Otherwise; \end{cases} \tag{10}$$

It should be noticed that the source node does not change during the fetching process. If all neighboring BSs of v_k do not cache the missed content j, the remote cloud is selected as the source node:

$$x_j^{t,(0,k)} = \mathbb{I}(\sum_{m=1}^{K} x_j^{t,(m,k)} == 0), r_0^{t,k} = \frac{BD}{\max\{\sum_j x_j^{t,(0,k)}, 1\}}; \tag{11}$$

Here, it can be found that the fetching rate of all links is calculated in real-time. When new fetching processes occur and compete for link resources, the fetching rate of old fetching processes decreases, thus increasing the fetching delay. When old fetching processes are resolved and the required contents arrive at the target BSs, the fetching rate of other fetching processes also increases, thus reducing the fetching delay.

Then, we can calculate $F_j^{\prime t,k}$ by subtracting the fetching size of the content j during the timeslot t from $F_j^{t,k}$:

$$F_j^{\prime t,k} = \max\{F_j^{t,k} - \sum_{m=0}^{K} x_j^{t,(m,k)} \times r_m^{t,k}, 0\}. \tag{12}$$

If there is no new fetching process triggered in case (2), $F_j^{t,k}$ can be directly determined according to the remaining size of the content j at the end of the timeslot $t-1$, $F_j^{'t-1,k}$:

$$F_j^{t,k}(old) = F_j^{'t-1,k}; \tag{13}$$

We use $F_j^{t,k}(old)$ to represent that the fetching process for content j is still in resolving. It should be noted that if there is no fetching process for content j, we can still use the equation to calculate the remaining size of the content since it always remains 0 until a new fetching process is triggered.

By combing the above two cases, $F_j^{t,k}$ can be determined by:

$$F_j^{t,k} = \max\{F_j^{t,k}(new), F_j^{t,k}(old)\}. \tag{14}$$

When $F_j^{t,k} > 0$ and $F_j^{'t,k} == 0$, the fetching process for the missed content j is successfully completed at the end of the timeslot t, which means that the missed content j arrives at v_k from the source node. We define the set of all newly arrived contents at the end of the timeslot t as:

$$O^{'t,k} = \bigcup_{F_j^{t,k} \times \mathbb{I}(F_j^{'t,k}==0)>0}^{1\leq j\leq C} \{j\}; \tag{15}$$

Because the content library is constant during the service term, only the contents that exist in the library can be fetched. So the range of j is between 1 and C.

After that, we use a cooperative cache replacement policy π to update the cache strategies of all BSs with their newly arrived contents:

$$\{U^{t+1,1}, U^{t+1,2}, ..., U^{t+1,K}\} = \pi(U^{t,1} \cup O^{'t,1}; U^{t,2} \cup O^{'t,2}; ...; U^{t,K} \cup O^{'t,K}). \tag{16}$$

4.3 Delivery Delay Model

The delivery delay model is built upon the fetching delay model. The BSs can only deliver contents to the users if the requests already exist on the BSs and their required contents arrive at the BSs, or if new requests arrive at the BSs and find their required contents already cached on the BSs. The BSs do not deliver contents to users if there are no requests from users or the contents required by requests are in the fetching processes. According to the above conditions, we set $Flag_{i,j}^{t,k}$ to judge whether v_k delivers the content j to the user i in the timeslot t:

$$Flag_{i,j}^{t,k} = \begin{cases} 1, & if \;\; (\mathbb{I}(j \in O^{'t-1,k}) \times d_{i,j}^{'t-1,k}) \vee (a_{i,j}^{t,k} \times \mathbb{I}(j \in U^{t,k})) == 1; \\ 0, & if \;\; d_{i,j}^{'t-1,k} + a_{i,j}^{t,k} \times \mathbb{I}(j \in U^{t,k}) == 0; \\ Flag_{i,j}^{t-1,k}, & otherwise; \end{cases} \tag{17}$$

Here, we use $V_{i,j}^{t,k}$ to record the remaining size of the content j from v_k to the user i in the timeslot t:

$$V_{i,j}^{t,k} = \begin{cases} 1, & if \;\; a_{i,j}^{t,k} == 1; \\ \max\{V_{i,j}^{t-1,k} - Flag_{i,j}^{t-1,k} \times r_i^{t-1,k}, 0\}, & otherwise. \end{cases} \tag{18}$$

The delivery rate $r_i^{t,k}$ from v_k to the user i can be calculated by $r_i^{t,k} = B\log_2(1 + SINR_i^{t,k})$ according to the Shannon capacity formula [31]. All wireless channels have the same bandwidth B, and $SINR_i^{t,k}$ is the signal-interference-noise ratio (SINR) from v_k to the user i in the timeslot t:

$$SINR_i^{t,k} = \frac{p_k \times |h_i^{k,t}|^2}{\sigma^2 + \sum_{u\neq i} |h_u^{k,t}|^2}, \tag{19}$$

where p_k is the transmit power of v_k, σ^2 is the power of background additive noise, and $|h_i^{k,t}|^2$ is the channel gain between v_k and the user i.

Then, we can deduce the value of $d'^{t,k}_{i,j}$ according to $V^{t,k}_{i,j}$:

$$d'^{t,k}_{i,j} = \begin{cases} 1, & if \;\; V^{t,k}_{i,j} - Flag^{t,k}_{i,j} \times r^{t,k}_i > 0; \\ 0, & otherwise. \end{cases} \tag{20}$$

4.4 Problem Formulation

This paper aims to find a cooperative cache replacement policy to minimize the transmission delay of requests in the network over a long term. From the system model, we can find that the transmission delay is affected by the subsequential requests and the dynamic channel features, so it is hard to calculate the transmission delay directly. To solve this issue, we count the number of requests in the network for each timeslot in real-time and obtain the total transmission delay of all requests during the service term by accumulating the request number for all timeslots. Therefore, we formulate the cooperative cache replacement problem as an optimization problem as follows:

$$\begin{aligned} &\min_{\pi} \sum_t \sum_k \sum_i \sum_j d^{t,k}_{i,j}, \\ s.t. \quad & C1: (3), (4), (6), (7), \\ & C2: U^{t,k} \cap O'^{t,k} = \emptyset, \forall t, k, \\ & C3: |U^{t,k}| \leq L, \forall t, k, \\ & C4: |U^{t,k}| + |O'^{t,k}| \leq C, \forall t, k, \\ & C5: \textstyle\sum_{m=0}^{K} x_j^{t,(m,k)} \leq 1, \forall t, k, j \end{aligned} \tag{21}$$

where constraint 2 ensures that the newly arrived contents of each BS are not already cached on the local BS; Constraint 3 guarantees that the number of cached contents does not exceed the storage capacity of each BS; In the cooperative edge caching-aided mobile edge computing network where a massive number of users simultaneously request different contents, the limited cached contents on the BSs are insufficient to meet the demands of such a substantial user base, necessitating continuous fetching of new contents from the cloud or neighboring BSs. Constraint 4 emphasizes that the sum of the number of cached content types and the number of the newly arrived content types is, at most, the number of cloud content types, which enforces that users can only request the contents that exist in the content library; Constraint 5 enforces that there only exists one source node for each fetching process on each BS. The main notations of the system model are presented in Table 1.

5 DECOR: A DELAY-AWARE COOPERATIVE CACHE REPLACEMENT METHOD BASED ON OFFLINE DRL

The delay-aware cooperative cache replacement method based on offline DRL named DECOR is proposed in this section in detail. First, we outline the DECOR system framework and explain how it works. Next, we model the cooperative cache replacement problem to an MDP model and discuss our design concept. Then, we present the structure of the Q-function neural network. Finally, the offline DRL algorithm is presented.

5.1 DECOR System Framework

The DECOR system framework is illustrated in Fig. 3, containing two components, an environment component and a training component. The environment component utilizes a behavior policy to constantly interact with the cooperative edge caching-aided MEC network to update the BSs' cache

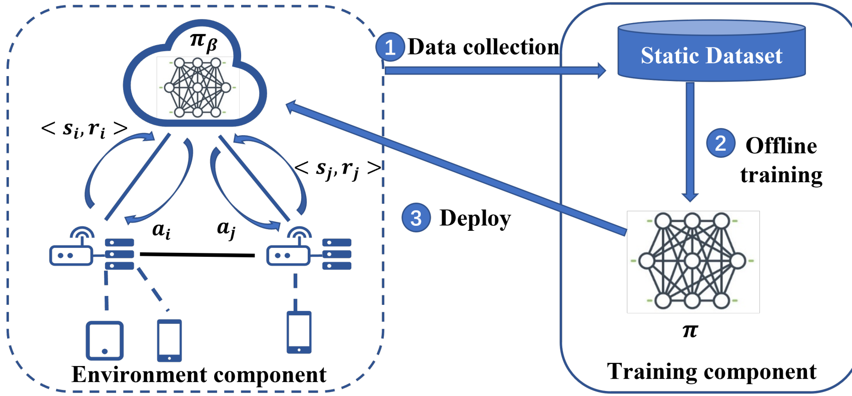


Fig. 3. DECOR System Framework. The environment component first samples a static dataset using the behavior policy π_β during execution. Then training component trains a policy offline π based on the dataset. In the training process, the training component does not interact with the environment component. Finally, the behavior policy π_β is replaced with the well-trained offline policy π.

strategies. The behavior policy is deployed in the cloud. In each timeslot, it receives the local states from all BSs and produces actions for them. Then the cloud sends the actions to the corresponding BSs. After the BSs execute these actions, they send new local states and rewards to the cloud. During this process, the cloud stores the states, actions, and rewards in a static dataset. The training component uses the static dataset to train an offline policy until the policy is well-trained. During the training process, the training component does not interact with the environment component, which means that even if the environment component samples new data, these data are not used in the training process [15]. At the same time, changes in the performance of the offline policy during the training process have no impact on the execution of the MEC network. After the offline policy is well-trained, it will replace the behavior policy to interact with the MEC network.

5.2 The Cooperative Edge Caching MDP Model

To apply the offline DRL to the cooperative edge caching problem, we should first model this problem as an MDP. We should carefully design the state space, action space, and reward function to ensure training convergence and improve the cache performance. Moreover, the behavior policy is not limited to DRL-based methods, it can also be heuristic-based or even rule-based methods (e.g., Least Recently Used (LRU), Least Frequently Used (LFU), First In First Out (FIFO)), which implies that the structure of the collected dataset may not meet the requirements of an MDP for the state, action, and reward. Therefore, the state space, action space, and reward function should be designed in a form that can be efficiently extracted from raw data. Furthermore, because the cloud should frequently communicate with the BSs to exchange states, actions, and rewards, they should be lightweight to reduce communication costs. Based on the above considerations, the state space, action space, and reward functions are defined as follows:

- **State Space:** Local states of all BSs construct the state observed by the decision-making policy $s_t = \{s_t^1; s_t^2; ...; s_t^k; ...s_t^K\}$. The local state design for each BS is similar. First, the local state should contain information that can reflect the content popularity because caching popular contents can efficiently avoid BSs frequently fetching contents from the cloud or the neighboring BSs, thus reducing the transmission delay. To reflect content popularity, we use the past requested content history as part of the local state s_t^k. We define the requested history

of cached contents as $\mathbf{X_t^k} = (\mathbf{x_t^{k,1}}, \mathbf{x_t^{k,2}}, ..., \mathbf{x_t^{k,l}}, ...\mathbf{x_t^{k,L}})$, where $\mathbf{x_t^{k,l}} = (x_t^{k,l}, x_{t-1}^{k,l}, ..., x_{t-n+1}^{k,l})$ is a vector that records the requested history of v_k in past n timeslots of the content cached in the lth location, and $x_t^{k,l}$ is the number of times the content cached in the lth location of v_K has been required by new requests in the timeslot t. Besides, the policy should also know the popularity of the newly arrived contents to make cache replacement decisions. Similar to the cached contents, we define the requested history of the newly arrived contents as $\mathbf{X_t^{'k}} = (\mathbf{x_t^{'k,1}}, \mathbf{x_t^{'k,2}}, ..., \mathbf{x_t^{'k,c}}, ...\mathbf{x_t^{'k,C}})$. Since the amount of new contents is variable which does not meet the fixed input size of DNN, we record the requested history of all contents instead of the new contents and set the value of $\mathbf{x_t^{'k,c}}$ as $\mathbf{0}$ vector, where $c \notin O'^{t,k}$. Apart from the content popularity, the fetching rate should be considered because it affects the arrival of missed contents, thus influencing the cache replacement decision-making. The remaining size of contents to be fetched reflects the information of the fetching rate to a certain, so we use its history $\mathbf{F_t^k} = (\mathbf{f_t^{k,1}}, \mathbf{f_t^{k,2}}, ..., \mathbf{f_t^{k,c}}, ...\mathbf{f_t^{k,C}})$ as part of the local state, where $\mathbf{f_t^{k,c}} = (F_c^{t,k}, F_c^{t-1,k}, ..., F_{cf}^{t-n+1,k})$. Similarly, because the number of the fetching process is variable, we record the history of all contents and set the value of $\mathbf{f_t^{k,c}}$ as 0 vector, where $F_c^{t,k} == 0$. Merging the parts of the state, We have $s_t = \{(\mathbf{X_t^1}, \mathbf{X_t^{'1}}, \mathbf{F_t^1}); (\mathbf{X_t^2}, \mathbf{X_t^{'2}}, \mathbf{F_t^2}); ...; (\mathbf{X_t^K}, \mathbf{X_t^{'K}}, \mathbf{F_t^K})\}$.

- **Action Space:** Local actions of all BSs construct the action produced by the policy $a_t = \{a_t^1; a_t^2; ...; a_t^k; ...a_t^K\}$. The local action of each BS has a similar design, and we use the design of a_t^k as an example: in the timeslot t, several missed contents $O'^{t,k}$ arrive at v_k. If $O'^{t,k} + U^{t,k} \leq L$, the missed contents can be directly cached into v_k. If $O'^{t,k} + U^{t,k} > L$, v_k should select some contents from $U^{t,k}$ to evict and choose the same number of conents from $O'^{t,k}$ to cache with the objective of minimizing the transmission delay. The eviction and cache steps can be combined into one step, i.e., selecting L contents from $O'^{t,k} \cup U^{t,k}$ to cache. In this case, the local action space size of a_t^k is a combination number $\mathbf{C}_L^{|O'^{t,k}|+|U^{t,k}|}$. However, the number of the newly arrived contents $O'^{t,k}$ is not fixed, but the output size of DNN is fixed, so we select L contents from the content library C instead of $O'^{t,k} \cup U^{t,k}$. In this way, the local action space size of a_t^k is fixed as $\mathbf{C}_L^C$.
- **Reward Function:** The cooperative cache replacement problem is formulated as an optimization problem to minimize the total transmission delay of all requests. However, the objective of DRL-based methods is to maximize the expected return. Thus, we define the negative value of the number of requests in the BS v_k per timeslot as the local reward $r_t^k(s_t^k, a_t^k) = \sum_i \sum_j -d_{i,j}^{t+1,k}$. Then, the reward function is the sum of the local reward of all BSs:

$$r_t(s_t, a_t) = \sum_k r_t^k(s_t^k, a_t^k) = \sum_k \sum_i \sum_j -d_{i,j}^{t+1,k}. \tag{22}$$

5.3 Neural Network Structure

According to the MDP model of the cooperative cache replacement problem, it can be easily found that the state is time series data. How to efficiently extract information about the content popularity, fetching rate, and their changing trends from the state needs to be considered. In addition, because we extend the action space size from $\mathbf{C}_L^{|O'^{t,k}|+|U^{t,k}|}$ to $\mathbf{C}_L^C$, there exists some invalid actions that cannot be executed in the MEC network. Therefore, how to filter invalid actions is another consideration.

For the first consideration, CNN has been proven to efficiently extract the temporary correlation features from time series data in recent years [6, 12], so we use CNN to extract knowledge about

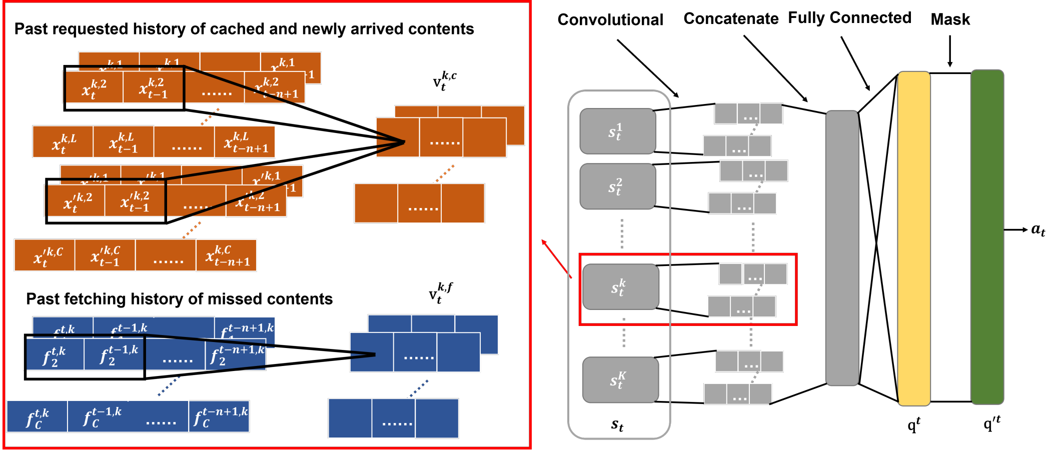


Fig. 4. Structure of the Q-function neural network

the content popularity and fetching rate from history. For the second consideration, we construct a mask layer to filter the invalid actions based on the information provided by the state. To be more specific, we construct a multi-head CNN, where each head processes the local state of one BS for extracting temporary knowledge about its content popularity and fetching rate. For any of the BS, its local state contains three parts: the requested history of cached contents, the requested history of newly arrived contents, and the fetching history of missed contents, where the requested history of cached and newly arrived contents is related to the content popularity and the fetching history of missed contents is related to the fetching rate. Due to the difference in the feature of the content popularity and fetching rate, each head uses two 1-D CNNs to extract their knowledge separately, as shown in Fig. 4. Thus, the number of 1-D CNNs in the Q-function neural network is $2 \times K = 2K$. For the local state s_t^k of the BS v_k, its corresponding 1-D CNNs are defined as $conv_c^k$ and $conv_f^k$. We also set the number of input channels for each CNN to be equal to the number of content types of its input, so the number of input channels for $conv_c^k$ is $L + C$, and the number of input channels for $conv_f^k$ is C. The temporary knowledge $\mathbf{v}_t^{k,c}$ and $\mathbf{v}_t^{k,f}$ are produced as follows:

$$\mathbf{v}_t^{k,c} = conv_c^k([\mathbf{X}_t^k, \mathbf{X}_t'^k]), \mathbf{v}_t^{k,f} = conv_f^k(\mathbf{F}_t^k). \tag{23}$$

To ensure that the local actions produced by the Q-function neural network are cooperative, it needs to aggregate the temporary knowledge of all BSs to capture a global view, then jointly produce the local actions for all BSs based on this view. Here, we concatenate the temporary knowledge of all BSs into a single vector as the input of a two-layer fully connected neural network (FCNN), defined as fc, to jointly produce the Q-value estimations for all actions:

$$\mathbf{q}^t = fc([\mathbf{v}_t^{1,c}, \mathbf{v}_t^{1,f}, \mathbf{v}_t^{2,c}, \mathbf{v}_t^{2,f}, \ldots, \mathbf{v}_t^{k,c}, \mathbf{v}_t^{k,f}, \ldots, \mathbf{v}_t^{K,c}, \mathbf{v}_t^{K,f}]). \tag{24}$$

We choose the action with the highest Q-value from the Q-value estimation $\mathbf{q}^t$ as the output. However, invalid actions are sometimes estimated with high Q-values, causing the policy to select invalid actions that cannot be executed. To filter these invalid actions, we construct a mask layer, defined as $mask_t$, according to the state s_t to set the Q-values of invalid actions as $-inf$ but not change the Q-values of valid actions. For any action $a_t = \{a_t^1; a_t^2; \ldots; a_t^k; \ldots; a_t^K\}$, its Q-value can be

updated by:

$$\mathbf{q}'^{\mathbf{t}}(a^t) = mask_t(\mathbf{q}^{\mathbf{t}}(a^t)) = \begin{cases} \mathbf{q}^{\mathbf{t}}(a^t), & if\ \prod_k a_t^k \in (O'^{t,k} \cup U^{t,k}); \\ -inf, & otherwise. \end{cases} \tag{25}$$

Then, we can calculate the probability of each action to be selected by inputting the Q-value estimation vector $\mathbf{q}^{\mathbf{t}}$ into a softmax layer:

$$prob(a^t) = \frac{e^{\mathbf{q}'^{\mathbf{t}}(a^t)}}{\sum_{\hat{a}^t} e^{\mathbf{q}'^{\mathbf{t}}(\hat{a}^t))}}; \tag{26}$$

The probability of invalid actions is $e^{-inf} = 0$, meaning that there is no chance of selecting invalid actions.

5.4 Offline DRL Algorithm

The Q-function can constantly fix its estimated Q-values for state-action pairs $< s_t, a_t >$ to approximate the true Q-values using the ground truth rewards of the newly sampled data according to Equation 2. Unfortunately, this approach does not work well in the offline scenario because only a static dataset, denoted as D, is available for training, and no new data is provided. As the offline policy is continuously updated, it will gradually deviate from the state access distribution of the dataset and explore some unseen state-action pairs outside the dataset. Because the offline policy cannot interact with the MEC network to sample these unseen state-action pairs, it lacks the ground truth rewards of the unseen pairs to fix their misestimated Q-values. These misestimated Q-values may guide the offline policy to select improper actions, thus degrading its performance. Moreover, using Equation 2, the effect of the misestimated Q-values may be widespread to other estimated Q-values, making the estimated Q-values of the seen state-action pairs wrong. The shift between the distribution of the offline policy and that of the dataset caused by the offline policy updates is usually regarded as a distribution shift. It is a critical challenge that needs to be addressed by the offline DRL.

Recently, many offline DRL algorithms have been developed to solve this challenge from different aspects. However, most of them have a core idea: to require that the state access distribution of the offline policy does not overly deviate from the state access distribution of the dataset. The authors in [8] use a conditional variational auto-encoder (VAE) to fit the distribution of the dataset and use a perturbation model to control the offset range between the distribution of VAE and that of the dataset. Advantage-Weighted Regression [21] adds KL-divergence into the loss function to constrain the offline policy from deviating too much from the distribution of the dataset. In this paper, we adopt an advanced offline DRL algorithm improved from CQL [14] to alleviate the distribution shift. We incorporate the working mechanism of Double DQN (DDQN) [26] into CQL to make the estimates of Q-values more conservative. The loss function of the improved CQL is presented as follows:

$$L_{CQL} = \dot{\alpha}(\mathbb{E}_{s_t \sim T^D, a_t \sim \pi(a_t|s_t)}[Q_\theta(s_t, a_t)] - \mathbb{E}_{s_t, a_t \sim T^D}[Q_\theta(s_t, a_t)]) + L_{DDQN}(\theta, \theta^-), \tag{27}$$

where T^D represents the distribution of the dataset. $L_{DDQN}(\theta, \theta^-)$ is the loss function of DDQN:

$$L_{DDQN}(\theta, \theta^-) = \frac{1}{2}\mathbb{E}_{s_t, a_t, r_t, s_{t+1} \sim T^D}[(Q_\theta(s_t, a_t) - (r_t + \gamma \hat{Q}_{\theta^-}(s_{t+1}, \arg\max_{a_{t+1}} Q_\theta(s_{t+1}, a_{t+1}))))^2]; \tag{28}$$

DDQN contains two neural networks: a Q-function network and a target Q-function network. It selects the next state-action pair $< s_{t+1}, a_{t+1} >$ depending on the highest Q-value estimated by the Q-function network, and uses the Q-values of the selected next state-action pair $< s_{t+1}, a_{t+1} >$

Algorithm 1 Offline DRL Algorithm

1: Collect a dataset D with a behavior policy π_β from the cooperative edge caching-aided MEC network.
2: Create a Q-function neural network Q with randomly generated initial parameters θ.
3: Create a target Q-function neural network $\hat{Q}$ with parameters $\theta^- = \theta$.
4: **for** step $t = 1, 2, 3, ..., N$ **do**
5: Randomly sample a minibatch of data from D to update Q_θ using G_Q gradient steps with Equation. 27:
$\theta_t \leftarrow \theta_{t-1} - \eta \bigtriangledown_\theta L_{CQL}$.
6: Every P steps reset $\hat{Q}_{\theta^-} = Q_\theta$.
7: **end for**

estimated by the target Q-function network to compute the loss and update the neural network parameters. Here is the loss function of DQN used in the original CQL:

$$L_{DQN}(\theta, \theta^-) = \frac{1}{2}\mathbb{E}_{s_t, a_t, r_t, s_{t+1} \sim T^D}[(Q_\theta(s_t, a_t) - (r_t + \gamma \max_{a_{t+1}} \hat{Q}_{\theta^-}(s_{t+1}, a_{t+1})))^2]; \tag{29}$$

It is obvious that the value of the second part of L_{DDQN} is closer or smaller than the value of the second part of L_{DQN} because the target Q-function network of DQN always selects the highest Q-value:

$$r_t + \gamma \hat{Q}_{\theta^-}(s_{t+1}, \arg\max_{a_{t+1}} Q_\theta(s_{t+1}, a_{t+1})) \le r_t + \gamma \max_{a_{t+1}} \hat{Q}_{\theta^-}(s_{t+1}, a_{t+1}); \tag{30}$$

Therefore, after several updates, the Q-function network tends to be more conservative in estimating Q-values. In this way, the estimated Q-values of the state-action pairs inside the dataset are closer to their true Q-values, and the estimated Q-values of the state-action pairs outside the dataset are more conservative to prevent the offline policy from exploring them. After that, we can find the optimal Q-function by minimizing the loss function of the improved CQL:

$$Q^*_\theta \leftarrow \arg\min_{Q_\theta} L_{CQL}. \tag{31}$$

The first part of L_{CQL} enforces that all state-action pairs are estimated with low Q-values. The second part of L_{CQL} encourages the Q-function network to give high Q-value estimates for the state-action pairs inside the dataset. The third part uses the background truth reward provided by the dataset to approximate the estimated Q-values to the true Q-values. By combining the three parts, L_{CQL} drives the Q-function network to select the state-action pairs available inside the dataset and avoid choosing the unseen state-action pairs outside the dataset. The hyperparameter α is a tradeoff factor used to adjust the weight of the first and second parts of L_{CQL}.

The details of the offline DRL algorithm are presented in Algorithm 1. At first, the behavior policy π_β deployed in the cloud constantly interacts with all BSs to collect a static dataset. It should be noticed that π_β can be derived from rule-based, heuristic-based, or DRL-based methods. Then, we initialize the parameters of the Q-function and target Q-function networks. After that, we constantly sample data from the dataset to compute the gradients according to Equation. 27. The gradients are used to update the parameters of the Q-function network. The update process will repeat multiple times until the Q-function network is well-trained. The parameters of the target Q-function network are assigned with the parameters of the Q-function network at every P steps. The hyperparameter η is the learning rate.

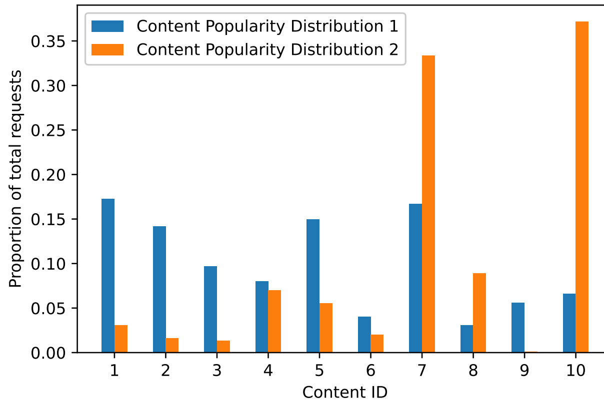


Fig. 5. The comparision between two distributions of the content popularity.

6 EXPERIMENT

This section evaluates the performance of DECOR and compares it to rule-based, deep learning (DL)-based, and DRL-based methods, where the rule-based method, LRU, is used as the behavior policy to sample static datasets:

- **LRU:** In each timeslot, LRU will replace the contents not requested for the longest time with the newly arrived contents. Each BS independently executes an LRU as its cache replacement policy.
- **LFU:** LFU will replace the least requested contents with the newly arrived contents.
- **FIFO:** FIFO will replace the earliest cached contents with the newly arrived contents.
- **Belady:** Belady [2] holds a view of future requests and will replace the contents not requested in the future for the longest time with newly arrived contents. Although Belady cannot be perfectly realized in the real world because we cannot obtain an accurate future view in the real environment, we can achieve a certain level of Belady by using DL to predict future requests. In this work, we use Belady as the upper bound of the rule-based methods to evaluate the superiority of our approach.
- **DL:** DL [16, 41] uses the states in the dataset as input and the actions in the dataset as outputs to learn their mapping relationship. Here, we use the categorical cross-entropy loss function to minimize the error between the given actions and the output actions so that the learned policy can be seen as a cloning policy of the behavior policy. DL does not utilize the rewards in the dataset.
- **DDQN:** The online DDQN [29] collects data by interacting with the environment and stores them in a replay buffer. Then, it samples a minibatch of data from the replay buffer to update the learned policy. After that, the new policy is used to collect new data. The above steps repeat until the policy is well-trained. We exchange the replay buffer with a static dataset and remove the data collection phase to apply DDQN in the offline scenario.

6.1 Simulation Setup

We design a MEC simulation environment including a remote cloud and $K = 2$ BSs. We set the size of 1 unit as 1 GB so that the storage size of the two BSs is $L = 3$ GB. The number of wired

backhaul links between the cloud and BSs is 5, and the link bandwidth is 256 MB/s. The number of the BS-to-BS links between the BSs and their bandwidth is set to 1 and 256 MB/s, respectively. For the wireless channel between the BSs and users, we set the transmit power of the BSs to $p_k = 46$ dBm, and the channel bandwidth is $B = 40$ MHz [4]. Recently, video service has become the major service in mobile networks [5], so we define the content type as video. There are a total of $C = 10$ video contents, and all contents have the same size of 1 GB.

We adopt a real-world dataset, Movielens [9], to simulate the request arrivals and user preferences by assuming that each movie is one content and each movie rating item corresponds to a request [19]. Thus, a user rates a movie at a specific time can be seen as a request arriving at the edge server for the corresponding content simultaneously. To simulate the explosion of user requests in the MEC network, we scale the time unit of Movielens from the hourly level to the second level. To avoid losing generality, the two BSs have different request arrivals and user preferences sampled from Movielens, shown in Fig. 5. The distribution of content popularity is calculated by:

$$Proportion^k(c) = \frac{\sum_t \sum_i a_{i,c}^{t,k}}{\sum_j \sum_t \sum_i a_{i,j}^{t,k}}; \tag{32}$$

The content popularity of all contents in distribution 1 is relatively close, which is quite different from distribution 2 where users prefer content 7 and 10 over others.

DECOR is based on a PyTorch implementation. The input channel number of 1-D CNN $conv_c$ is $C + L = 13$, and $conv_f$ is $C = 10$ for each BS. Both CNNs have 2 kernel sizes and 64 output channels, followed by a 1-D MaxPool layer with padding size 2. Each layer of the two-layer FCNN fc has 64 neurons. We choose ReLU as the activation function due to its fast updating speed. Moreover, the learning rate of offline training is set to 1×10^{-6}. The target Q-function network update interval is set to $P = 30$, and the minibatch size is 128. The parameters of the Q-function network are optimized via Adam [13]. The history length of states is $n = 10$ s.

6.2 Results analysis

We first test the offline training convergence of DECOR, and its learning curve is shown in Fig. 6. It can be found that although DECOR fluctuates between the training epochs 51 and 72, it still successfully converges to a better point than the behavior policy, LRU, with a delay reduction rate of 33%. In contrast, the online DRL-based method, DDQN, fails to learn a better cooperative cache policy from the static dataset and even converges to a worse point than LRU. Besides, since the optimization objective of DL is the distribution divergence between its policy and the dataset, the DL's curve only shows the cache performance of its policy. When DL converges, its learned policy can be seen as a cloning policy for the dataset and has a similar cache performance to LRU. Thus, it cannot learn a better policy from the dataset either. Fig. 7 compares the cache performance of DECOR to the rule-based, DL-based, and DRL-based methods. It can be found that DECOR outperforms all other methods, which shows its superiority. Although Belady maintains a future view with a length of 48 s, since its cache replacement rule is quite simple, it still loses to DECOR.

Moreover, we investigate the impact of two resource factors, the storage size of BSs and the wired link bandwidth that are closely related to the cooperative cache replacement policy, on the cache performance, which is presented in Fig. 8. Since DDQN cannot learn a better policy from the dataset and DL just learns a cloning policy, we do not evaluate their cache performance. In general, the average transmission delay of all methods decreases in a rapid and then falt trend as both two resources increase. It is not hard to understand because when resources are sufficient, the most popular contents are cached on the BSs, or the missed contents can be quickly fetched from the neighboring BSs or the cloud. In this case, the fetching delay is very short, even if no cache

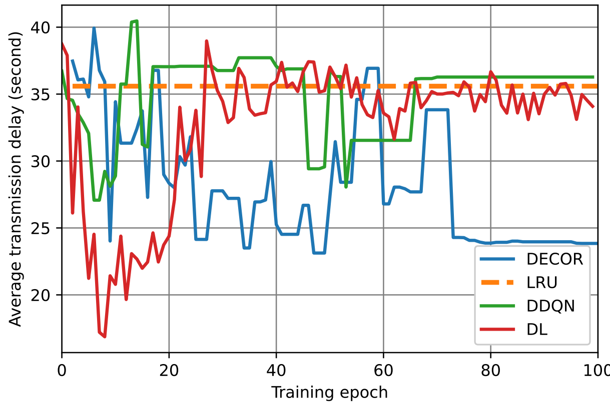


Fig. 6. The learning curve comparision between DL, DDQN and DECOR.

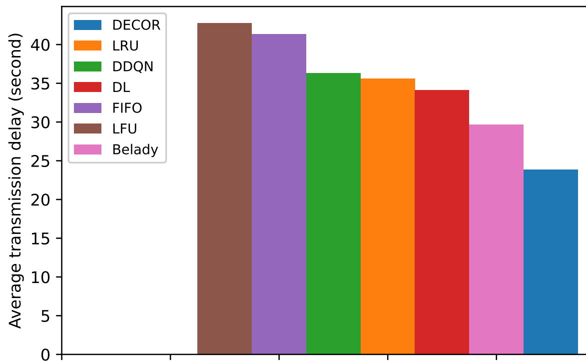
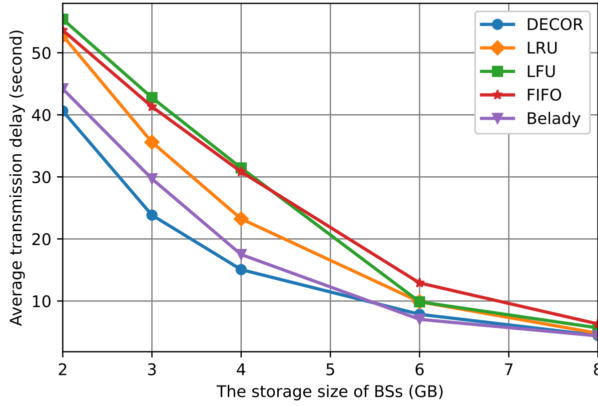


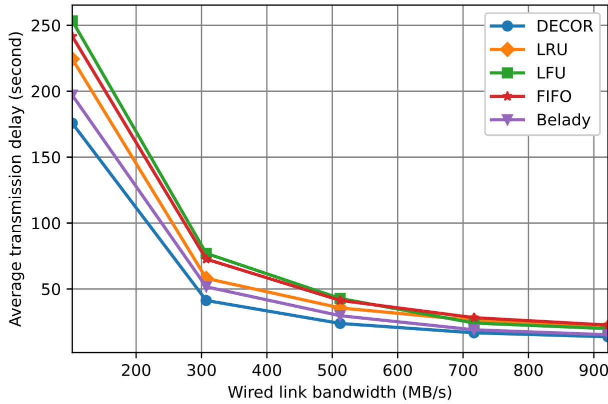
Fig. 7. The cache performance comparision.

replacement policy is implemented. In Fig. 8(a), DECOR outperforms all rule-based methods in most cases, except when the storage size is 6 and 8 GB, which is inferior to Belady. The reason is when the storage size is 6 and 8 GB, the most popular contents are cached, and only a small number of unpopular contents need to be replaced. Belady is suitable for the case because it replaces the contents not requested in the future for the longest time. In Fig. 8(b), we find that the improvement rate of DECOR to LRU is 21.7%, 28.9%, 33%, 35.9%, and 34.6% as the wired link bandwidth increases, showing a trend of growth followed by decline. This is because the fetching process becomes shorter when the bandwidth increases, resulting in fewer subsequential requests that can share it. In this case, BSs need to fetch contents to replace the cached ones more frequently, which strengthens the effect of the cooperative cache replacement policy. However, when the bandwidth is sufficient, the fetching delay is reduced to 0, and the cache replacement policy has no effect.

Finally, we verify whether the cache replacement policy learned by DECOR can coordinate the two BSs. Here, we test the delay reduction rate of DECOR to LRU on the average transmission



(a)



(b)

Fig. 8. The impact of storage size and wired link bandwidth on the cache performance.

delay in three cases: (i) the two BSs have different content popularity distributions 1 and 2 shown in Fig. 5, respectively, (ii) they have the same content popularity distribution 1, (iii) they have the same content popularity distribution 2. Fig. 9 shows that the delay reduction rate in three cases is 33%, 37.6%, and 46.4%. It shows that the cache replacement policy of DECOR can utilize the storage of BSs more effectively when the two BSs have the same content popularity distribution. This is because when the two BSs have different distributions, the cache replacement policy is hard to coordinate the BSs to cache one content that is popular in one BS but unpopular in another. However, when the two BSs have the same distribution, the policy can better coordinate them to achieve a higher delay reduction rate, which verifies that the policy is cooperative. Besides, we find that the delay reduction rate in case (ii) is lower than in case (iii). The reason is that the

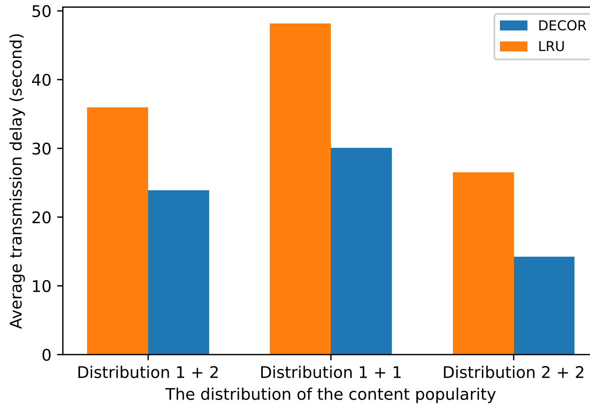


Fig. 9. The impact of content popularity on the cache performance.

content popularity of all contents in distribution 1 is close, thus degrading the effect of the cache replacement policy.

7 CONCLUSION

This paper has proposed a novel delay-aware cooperative cache replacement method based on offline DRL, named DECOR, to coordinate the cache replacement decisions of different edge servers for high cache performance in an MEC network of the smart city. Different from the existing online DRL-based methods, DECOR can effectively exploit the potential of the huge data in the smart city and avoids the learned policy at the early stage of the training process degrading the cache performance of the actual network. To efficiently extract the knowledge about content popularity and content fetching history from time series data, CNNs are utilized to improve training efficiency and cache performance. The experimental results show that DECOR can successfully learn an offline policy from a static dataset, compared to an online DRL-based method. The offline policy outperforms the behavior policy used to sample the dataset and can coordinate edge servers to make cooperative cache replacement decisions.

REFERENCES

[1] Nirav Atre, Justine Sherry, Weina Wang, and Daniel S. Berger. 2020. Caching with Delayed Hits. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication.* ACM, 495–513. https://doi.org/10.1145/3387514.3405883

[2] L. A. Belady. 1966. A study of replacement algorithms for a virtual-storage computer. *IBM Systems Journal* 5, 2 (1966), 78–101. https://doi.org/10.1147/sj.52.0078

[3] Shuangwu Chen, Zhen Yao, Xiaofeng Jiang, Jian Yang, and Lajos Hanzo. 2021. Multi-Agent Deep Reinforcement Learning-Based Cooperative Edge Caching for Ultra-Dense Next-Generation Networks. *IEEE Transactions on Communications* 69, 4 (2021), 2441–2456. https://doi.org/10.1109/TCOMM.2020.3044298

[4] Tzu-Yu Chen, Chih-Hang Wang, Jang-Ping Sheu, Guang-Siang Lee, and De-Nian Yang. 2022. Resource Allocation for the 4G and 5G Dual-Connectivity Network with NOMA and NR. In *ICC 2022 - IEEE International Conference on Communications.* 3784–3789. https://doi.org/10.1109/ICC45855.2022.9838985

[5] Ericsson. 2022. Ericsson Mobility Report. (2022). https://www.ericsson.com/en/reports-and-papers/mobility-report/reports/june-2022

[6] Hassan Ismail Fawaz, Benjamin Lucas, Germain Forestier, Charlotte Pelletier, Daniel F. Schmidt, Jonathan Weber, Geoffrey I. Webb, Lhassane Idoumghar, Pierre-Alain Muller, and FranÃğois Petitjean. 2020. InceptionTime: Finding

AlexNet for time series classification. *Data Mining and Knowledge Discovery* 34 (2020), 1936–1962. https://doi.org/10.1007/s10618-020-00710-y

[7] Amal Feriani and Ekram Hossain. 2021. Single and Multi-Agent Deep Reinforcement Learning for AI-Enabled Wireless Networks: A Tutorial. *IEEE Communications Surveys & Tutorials* 23, 2 (2021), 1226–1252. https://doi.org/10.1109/COMST.2021.3063822

[8] Scott Fujimoto, David Meger, and Doina Precup. 2019. Off-Policy Deep Reinforcement Learning without Exploration. In *Proceedings of the 36th International Conference on Machine Learning*, Vol. 97. PMLR, 2052–2062. https://proceedings.mlr.press/v97/fujimoto19a.html

[9] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. *ACM Transactions on Interactive Intelligent Systems* 5, 4 (2015), 1–19. https://doi.org/10.1145/2827872

[10] Syed Qaisar Jalil, Stephan Chalup, and Mubashir Husain Rehmani. 2021. Cognitive Radio Spectrum Sensing and Prediction Using Deep Reinforcement Learning. In *2021 International Joint Conference on Neural Networks (IJCNN)*. 1–8. https://doi.org/10.1109/IJCNN52387.2021.9533497

[11] Wenpeng Jing, Xiangming Wen, Zhaoming Lu, and Haijun Zhang. 2019. User-Centric Delay-Aware Joint Caching and User Association Optimization in Cache-Enabled Wireless Networks. *IEEE Access* 7 (2019), 74961–74972. https://doi.org/10.1109/ACCESS.2019.2918334

[12] Kathan Kashiparekh, Jyoti Narwariya, Pankaj Malhotra, Lovekesh Vig, and Gautam Shroff. 2019. ConvTimeNet: A Pre-trained Deep Convolutional Neural Network for Time Series Classification. In *Proceedings of the 2019 International Joint Conference on Neural Networks*. IEEE, 1–8. https://doi.org/10.1109/IJCNN.2019.8852105

[13] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations*. http://arxiv.org/abs/1412.6980

[14] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. 2020. Conservative Q-Learning for Offline Reinforcement Learning. In *Advances in Neural Information Processing Systems*, Vol. 33. Curran Associates, Inc., 1179–1191. https://proceedings.neurips.cc/paper/2020/file/0d2b2061826a5df3221116a5085a6052-Paper.pdf

[15] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. 2020. Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems. (2020). https://arxiv.org/abs/2005.01643

[16] Evan Liu, Milad Hashemi, Kevin Swersky, Parthasarathy Ranganathan, and Junwhan Ahn. 2020. An Imitation Learning Approach for Cache Replacement. In *Proceedings of the 37th International Conference on Machine Learning*, Vol. 119. PMLR, 6237–6247. https://proceedings.mlr.press/v119/liu20f.html

[17] Tang Liu, Baijun Wu, Wenzheng Xu, Xianbo Cao, Jian Peng, and Hongyi Wu. 2021. RLC: A Reinforcement Learning-Based Charging Algorithm for Mobile Devices. *ACM Transactions on Sensor Networks* 17, 4 (2021), 1–23. https://doi.org/10.1145/3453682

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533. https://doi.org/10.1038/nature14236

[19] Sabrina MÃijller, Onur Atan, Mihaela van der Schaar, and Anja Klein. 2017. Context-Aware Proactive Content Caching With Service Differentiation in Wireless Networks. *IEEE Transactions on Wireless Communications* 16, 2 (2017), 1024–1036. https://doi.org/10.1109/TWC.2016.2636139

[20] Dinithi Nallaperuma, Rashmika Nawaratne, Tharindu Bandaragoda, Achini Adikari, Su Nguyen, Thimal Kempitiya, Daswin De Silva, Damminda Alahakoon, and Dakshan Pothuhera. 2019. Online Incremental Machine Learning Platform for Big Data-Driven Smart Traffic Management. *IEEE Transactions on Intelligent Transportation Systems* 20, 12 (2019), 4679–4690. https://doi.org/10.1109/TITS.2019.2924883

[21] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. 2019. Advantage-Weighted Regression: Simple and Scalable Off-Policy Reinforcement Learning. (2019). https://arxiv.org/abs/1910.00177

[22] Rafael Figueiredo Prudencio, Marcos R. O. A. Maximo, and Esther Luna Colombini. 2023. A Survey on Offline Reinforcement Learning: Taxonomy, Review, and Open Problems. *IEEE Transactions on Neural Networks and Learning Systems* (2023). https://doi.org/10.1109/TNNLS.2023.3250269 Early Access.

[23] Shihao Shen, Yiwen Han, Xiaofei Wang, and Yan Wang. 2019. Computation Offloading with Multiple Agents in Edge-Computing-Supported IoT. *ACM Transactions on Sensor Networks* 16, 1 (2019), 1–27. https://doi.org/10.1145/3372025

[24] Junaid Shuja, Kashif Bilal, Waleed Alasmary, Hassan Sinky, and Eisa Alanazi. 2021. Applying machine learning techniques for caching in next-generation edge networks: A comprehensive survey. *Journal of Network and Computer Applications* 181 (2021), 103005. https://doi.org/10.1016/j.jnca.2021.103005

[25] Chunhe Song, Wenxiang Xu, Tingting Wu, Shimao Yu, Peng Zeng, and Ning Zhang. 2021. QoE-Driven Edge Caching in Vehicle Networks Based on Deep Reinforcement Learning. *IEEE Transactions on Vehicular Technology* 70, 6 (2021), 5286–5295. https://doi.org/10.1109/TVT.2021.3077072

[26] Hado van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double Q-Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 30. https://doi.org/10.1609/aaai.v30i1.10295

[27] Fangxin Wang, Feng Wang, Jiangchuan Liu, Ryan Shea, and Lifeng Sun. 2020. Intelligent Video Caching at Network Edge: A Multi-Agent Deep Reinforcement Learning Approach. In *IEEE Conference on Computer Communications*. IEEE, 2499–2508. https://doi.org/10.1109/INFOCOM41043.2020.9155373

[28] Ning Wang, Gangxiang Shen, Sanjay Kumar Bose, and Weidong Shao. 2019. Zone-Based Cooperative Content Caching and Delivery for Radio Access Network With Mobile Edge Computing. *IEEE Access* 7 (2019), 4031–4044. https://doi.org/10.1109/ACCESS.2018.2888602

[29] Xiaofei Wang, Yiwen Han, Chenyang Wang, Qiyang Zhao, Xu Chen, and Min Chen. 2019. In-Edge AI: Intelligentizing Mobile Edge Computing, Caching and Communication by Federated Learning. *IEEE Network* 33, 5 (2019), 156–165. https://doi.org/10.1109/MNET.2019.1800286

[30] Xiaofei Wang, Ruibin Li, Chenyang Wang, Xiuhua Li, Tarik Taleb, and Victor C. M. Leung. 2021. Attention-Weighted Federated Deep Reinforcement Learning for Device-to-Device Assisted Heterogeneous Collaborative Edge Caching. *IEEE Journal on Selected Areas in Communications* 39, 1 (2021), 154–169. https://doi.org/10.1109/JSAC.2020.3036946

[31] Xiaofei Wang, Chenyang Wang, Xiuhua Li, Victor C. M. Leung, and Tarik Taleb. 2020. Federated Deep Reinforcement Learning for Internet of Things With Decentralized Cooperative Edge Caching. *IEEE Internet of Things Journal* 7 (2020), 9441–9455. https://doi.org/10.1109/JIOT.2020.2986803

[32] Wanli Wen, Ying Cui, Fu-Chun Zheng, Shi Jin, and Yanxiang Jiang. 2018. Random Caching Based Cooperative Transmission in Heterogeneous Wireless Networks. *IEEE Transactions on Communications* 66, 7 (2018), 2809–2825. https://doi.org/10.1109/TCOMM.2018.2808188

[33] Teng Xiao and Donglin Wang. 2021. A General Offline Reinforcement Learning Framework for Interactive Recommendation. *Proceedings of the AAAI Conference on Artificial Intelligence* 35, 5 (May 2021), 4512–4520. https://doi.org/10.1609/aaai.v35i5.16579

[34] Dianlei Xu, Tong Li, Yong Li, Xiang Su, Sasu Tarkoma, Tao Jiang, Jon Crowcroft, and Pan Hui. 2020. Edge Intelligence: Architectures, Challenges, and Applications. (2020). https://arxiv.org/abs/2003.12172

[35] Siya Xu, Xin Liu, Shaoyong Guo, Xuesong Qiu, and Luoming Meng. 2021. MECC: A Mobile Edge Collaborative Caching Framework Empowered by Deep Reinforcement Learning. *IEEE Network* 35, 4 (2021), 176–183. https://doi.org/10.1109/MNET.011.2000663

[36] Jingjing Yao, Tao Han, and Nirwan Ansari. 2019. On Mobile Edge Caching. *IEEE Communications Surveys & Tutorials* 21, 3 (2019), 2525–2553. https://doi.org/10.1109/COMST.2019.2908280

[37] Xianyuan Zhan, Haoran Xu, Yue Zhang, Xiangyu Zhu, Honglei Yin, and Yu Zheng. 2022. DeepThermal: Combustion Optimization for Thermal Power Generating Units Using Offline Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence* 36, 4 (Jun. 2022), 4680–4688. https://doi.org/10.1609/aaai.v36i4.20393

[38] Ticao Zhang and Shiwen Mao. 2019. Cooperative Caching for Scalable Video Transmissions Over Heterogeneous Networks. *IEEE Networking Letters* 7, 2 (2019), 63–67. https://doi.org/10.1109/LNET.2019.2911972

[39] Xu Zhang, Zhengnan Qi, Geyong Min, Wang Miao, Qilin Fan, and Zhan Ma. 2022. Cooperative Edge Caching Based on Temporal Convolutional Networks. *IEEE Transactions on Parallel and Distributed Systems* 33, 9 (2022), 2093–2105. https://doi.org/10.1109/TPDS.2021.3135257

[40] Xiaoyan Zhao, Peiyan Yuan, Haiwen li, and Shaojie Tang. 2018. Collaborative Edge Caching in Context-Aware Device-to-Device Networks. *IEEE Transactions on Vehicular Technology* 67, 10 (2018), 9583–9596. https://doi.org/10.1109/TVT.2018.2858254

[41] Chen Zhong, M. Cenk Gursoy, and Senem Velipasalar. 2018. A deep reinforcement learning-based framework for content caching. In *2018 52nd Annual Conference on Information Sciences and Systems (CISS)*. 1–6. https://doi.org/10.1109/CISS.2018.8362276

[42] Xiaokang Zhou, Yiyong Hu, Jiayi Wu, Wei Liang, Jianhua Ma, and Qun Jin. 2023. Distribution Bias Aware Collaborative Generative Adversarial Network for Imbalanced Deep Learning in Industrial IoT. *IEEE Transactions on Industrial Informatics* 19, 1 (2023), 570–580. https://doi.org/10.1109/TII.2022.3170149

[43] Xiaokang Zhou, Wei Liang, Kevin I-Kai Wang, Zheng Yan, Laurence T. Yang, Wei Wei, Jianhua Ma, and Qun Jin. 2023. Decentralized P2P Federated Learning for Privacy-Preserving and Resilient Mobile Robotic Systems. *IEEE Wireless Communications* 30, 2 (2023), 82–89. https://doi.org/10.1109/MWC.004.2200381

[44] Xiaokang Zhou, Wei Liang, Kevin I-Kai Wang, and Laurence T. Yang. 2021. Deep Correlation Mining Based on Hierarchical Hybrid Networks for Heterogeneous Big Data Recommendations. *IEEE Transactions on Computational Social Systems* 8, 1 (2021), 171–178. https://doi.org/10.1109/TCSS.2020.2987846

[45] Xiaokang Zhou, Wei Liang, Ke Yan, Weimin Li, Kevin I-Kai Wang, Jianhua Ma, and Qun Jin. 2023. Edge-Enabled Two-Stage Scheduling Based on Deep Reinforcement Learning for Internet of Everything. *IEEE Internet of Things Journal* 10, 4 (2023), 3295–3304. https://doi.org/10.1109/JIOT.2022.3179231

[46] Xiaoping Zhou, Zhenlong Liu, Maozu Guo, Jichao Zhao, and Jialin Wang. 2022. SACC: A Size Adaptive Content Caching Algorithm in Fog/Edge Computing Using Deep Reinforcement Learning. *IEEE Transactions on Emerging Topics in Computing* 10, 4 (2022), 1810–1820. https://doi.org/10.1109/TETC.2021.3115793

[47] Xiaokang Zhou, Xiaozhou Ye, Kevin I-Kai Wang, Wei Liang, Nirmal Kumar C. Nair, Shohei Shimizu, Zheng Yan, and Qun Jin. 2023. Hierarchical Federated Learning With Social Context Clustering-Based Participant Selection for Internet of Medical Things Applications. *IEEE Transactions on Computational Social Systems* (2023). https://doi.org/10.1109/TCSS.2023.3259431 Early Access.

Received 20 December 2022; revised 23 July 2023; accepted 27 August 2023