Double Deep Q-Network based Joint Edge Caching and Content Recommendation with Inconsistent File Sizes in Fog-RANs

Jie Yan, Min Zhang, Yanxiang Jiang*, *Senior Member, IEEE*, Fu-Chun Zheng*, *Senior Member, IEEE*, Qi Chang, Khamael M. Abualnaja, Shahid Mumtaz, *Senior Member, IEEE*, and Xiaohu You, *Fellow, IEEE*

***Abstract*—In this paper, the joint edge caching and content recommendation problem for the case of inconsistent file sizes is investigated for fog radio access networks (F-RANs). Firstly, we transform the joint caching and recommendation policy into a 'single' caching policy. Then, a time-varying personalized user request model is proposed to describe the fluctuant demands of users. To maximize the long-term net profit of each fog access point (F-AP), we formulate the caching optimization problem and resort to a reinforcement learning (RL) framework. To address the impact of file size inconsistency, a 'pre-split' mechanism with a dynamic upper limit is adopted to meet the constraint of storage capacity, and a 'lazy' updating mechanism is introduced into the training process. Finally, a double deep Q-network (DDQN) based distributed edge caching algorithm is proposed with content recommendation. Simulation results show that compared with the existing methods, the average net profit of our proposed algorithm can be increased up to 29.7% while content recommendation can not only increase caching efficiency but also accelerate convergence.**

***Index Terms*—Fog radio access networks, deep reinforcement learning, edge caching, content recommendation.**

I. Introduction

The rapid popularization of intelligent devices brings a huge traffic pressure to wireless networks in recent years [1]–[4]. A new network architecture called fog radio access networks (F-RANs) has become a promising solution in industry and academia, because it puts popular contents from the cloud server to the network edge, which can bring better performance [5]–[8]. In F-RANs, some nodes at the edge of networks are designated as fog access points (F-APs), equipped with limited computing and caching resources. By considering dynamic requests of users and the constraint of storage capacity, how, when and what to cache strategically are three core requirements for each F-AP to obtain a higher caching efficiency [9]–[11]. There are many traditional distributed caching methods for optimal caching policy to improve retrieval robustness, delivery latency and data availability [12]–[16], or to solve caching fairness problem [4], such as belief propagation-based method [12], effective graph-based method [13], [14], alternating direction method of multipliers (ADMM) [15], mean field game-based method [16] and connected facility location approximation algorithm [4]. However, these existing traditional caching schemes have all assumed that the content popularity is known in advance and may remain constant for a long period. This, of course, may not be true in practice.

How to obtain these content popularity information in advance is indeed one of the critical issues in caching systems. Recently, the learning-based methods have been utilized to characterize the users' preference and predict the content popularity for the purpose of finding the optimal caching strategy [17]–[19]. Reinforcement learning (RL) is one effective solution to find the optimal caching policy for the case of unknown content popularity in a model-free fashion [20]–[25], which requires no prior knowledge on file popularity. RL has the ability to maximize the expected cumulative reward of resource consumption without the prior knowledge of the considered network. The approach is self-adaptive in dynamic environments. In [20], the hidden Markov decision process was applied to characterize the user request model, and then a Q-learning algorithm was used to establish a learning framework which can optimize caching strategy in a distributed manner. In [22], the authors optimized content caching at base stations (BSs) with unknown prior knowledge of user preference after personalized recommendation by an ϵ-greedy algorithm. In [23], an asynchronous caching scheme with adaptive parameters adjusted the caching policy to cope with the spatiotemporal variation of file popularity by light-weight updates. In [24], the authors designed a joint computing and caching framework by integrating deep deterministic policy gradient (DDPG) algorithm to minimize energy cost of mobile network operators (MNOs). In [25], a deep reinforcement learning-based resource allocation scheme was proposed to improve content distribution in a layered F-RAN, and the optimal resource allocation problem was formulated as a minimal delay model. These above works [17], [18], [20],

Corresponding authors: Yanxiang Jiang and Fu-Chun Zheng.

Manuscript received November 10, 2022; revised April 7, 2023, August 19, 2023; and accepted October 24, 2023. This work was supported in part the National Key Research and Development Program under Grant 2021YFB2900300, and by the National Natural Science Foundation of China under grant 61971129.

J. Yan, M. Zhang Y. Jiang, Q. Chang, and X. You are with the National Mobile Communications Research Laboratory, Southeast University, Nanjing 210096, China (e-mail: yanjied13@163.com, {mzhing, yxjiang, qchang, xhyu}@seu.edu.cn).

F. Zheng is with the School of Electronic and Information Engineering, Harbin Institute of Technology, Shenzhen 518055, China, and the National Mobile Communications Research Laboratory, Southeast University, Nanjing 210096, China (e-mail: fzheng@ieee.org).

K. Abualnaja is with College of Science, Taif University, Taif 21944, Saudi Arabia (e-mail: k.ala@tu.edu.sa).

S. Mumtaz is with the Department of Applied Informatics, Silesian University of Technology Akademicka 16 44-100 Gliwice, Poland and Nottingham Trent University, Departement of Engineering. (e-mail: dr.shahid.mumtaz@ieee.org).

[22]–[25] are all based on machine learning scheme to predict user request or file popularity without any prior knowledge. Considering the specificity and randomness of user requests, the difficulty of convergence and the volatility of the algorithm are correspondingly high, and the prediction accuracy is correspondingly low. In addition, most of the above studies assume that the files to be cached have uniform sizes, which is not consistent with the real caching scenario.

Since user requests are influenced by content recommendation, the heterogeneity of user requests could be reduced by appropriate content recommendation, and the difficulty of user request prediction or file popularity prediction can then be reduced accordingly [26]. As such, in recent years, content recommendation has been widely combined with wireless caching schemes. In [27], a joint pushing and recommendation scheme of multi-user multi-input single-output system was designed. The scheme pushes content items through multi-user multi-input single-output downlink and adopts a multicast beamforming method, which has the potential of significantly improving hit ratio due to joint caching and recommendation. In [28], an RL scheme was applied to the caching strategy for BSs under the influence of recommendation. The policies considered in [27] and [28], however, ignored the fact that user mobility affects user requests, which may result in deviations. In [29], a heuristic algorithm was employed to determine the contents of the cache and the recommendations for each user to maximize the cache hit ratio under the maximum tolerable distortion of the recommendations. In [30], the authors decoupled the joint problem of personalized recommendation into two RL sub-problems for the sake of circumventing the curse of dimensionality. Similarly, in [31], [32], the authors focused on the spatial-temporal recommendation, and a BS caching policy based on deep reinforcement learning (DRL) was considered to improve cache efficiency. However, in multi-user scenarios, the joint optimization problems can incur high resource consumption and the performance of the personalized recommendation schemes is often not satisfactory [29]–[32].

Most research in file caching leverages historical user request data to optimize the placement process of the cached files [33]–[35]. These caching methods cache the files independently and are called uncoded caching. On the other hand, the caching efficiency can be further improved by multicasting a combination of the requested files during the delivery process, which is called encoded caching [36]–[38]. However, some existing studies involving uncoded caching have assumed that the files in the cloud server have uniform size [23], [39], [40]. Although this assumption can greatly simplify the caching policy optimization, it might not reflect the real status of files, because files may well have non-uniform sizes in the cloud server. In addition, for large files, this assumption of uniform size may lead to other caching problems. For example, the transfer time and caching cost are obviously different between large files and small files.

The above discussions indicate that it is essential to deal with the edge caching problem by considering content recommendation while there should be no limits on the file sizes in the cloud server. In view of this, in this paper, we investigate edge caching policy to cope with dynamic user request with content recommendation in F-RANs to maximize the long-term net profit of each F-AP. Here, the net profit is the fee charged by a MNO for user requests minus the total transmission cost incurred during the communication process. The main contributions are summarized below.

1) We introduce content recommendations into the edge caching policy. In order to reduce the excessive resource consumption caused by joint optimization in multi-user scenarios, the recommendation policy is incorporated into the caching policy, where the 'abstract' parts of its current cached files are recommended by each F-AP to users. We formulate the caching optimization problem firstly and then solve it via an RL framework. And considering the accumulation property of action-value function of the RL framework, we define the optimization objective as maximizing the long-term net profit of each F-AP.
2) Considering that existing datasets of user requests is not compatible with the above content recommendation policy, we propose a time-varying personalized user request model to characterize the dynamic user requests after content recommendation. This user request model forms the external environment of the RL framework.
3) Different from some existing studies involving edge caching which assumed that the files stored in the cloud server have the uniform size, we impose no limits on the sizes of files in the cloud server. And to solve the problem caused by non-uniform file sizes, a 'pre-split' procedure with a dynamic upper limit is introduced to the cached files to meet the constraint of F-AP's limited storage capacity. At the same time, in order to match the dynamic 'pre-split' mechanism, a 'lazy' updating step is added into the training of the relevant parameters in the RL framework.
4) An edge caching algorithm based on DDQN with content recommendation is proposed. Different from some existing traditional caching schemes which need to know the prior knowledge of content popularity in advance, this algorithm, as an RL algorithm, requires no prior knowledge on file popularity, but optimizes it by interacting with the dynamic user requests constructed by the proposed request model.

The rest of this paper is organized as follows. In Section II, the system model is introduced. The proposed time-varying personalized user request model is described in Section III elaborately. In Section IV, the problem formulation and the proposed DDQN-based caching algorithm are presented. Simulation results are shown in Section V. Final conclusion is drawn in Section VI.

II. System Model

The edge caching problem with dynamic content recommendation is considered in a given area served by F-RANs, where H F-APs are covered by the cloud server, as shown in Fig. 1. The caching updates can be made according to the caching policy in discrete time slots denoted by $t = 0, 1, \ldots, T$. In our system model, there are files of

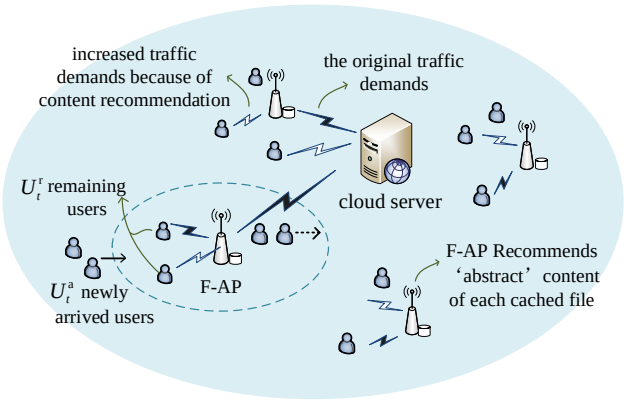


Fig. 1. Illustration of caching scenario with content recommendation in F-RANs.

different sizes in the cloud server. Let $S_{\max,t}$ denote the maximum size of the cached files in each F-AP during time slot t, which limits the maximum capacity occupied by each cached file to keep the total capacity from exceeding the storage capacity of the F-AP when the cached files need to be updated. For large files such as videos, users tend to enjoy them from the beginning. When the size of the file to be cached is greater than $S_{\max,t}$, the F-AP can only cache the front portion of the file with the size $S_{\max,t}$, i.e., the large files need to be 'pre-split'. This assumption can ensure the maximum diversity of cached files in the current F-AP. We assume that the caching capacity C_{f} of each F-AP is uniform and can cache $F_t = C_{\mathrm{f}}/S_{\max,t}$ files during time slot t. Let $\mathcal{C} = \{1, 2, \ldots, c, \ldots, C\}$ denote the file library and $\mathcal{S} = \{S_1, \ldots, S_C\}$ denote the set of file sizes in the cloud server. In addition, let $\mathcal{S}_t^{\mathrm{c}} = \{S_{t,i_1}^{\mathrm{c}}, \ldots, S_{t,i_f}^{\mathrm{c}}, \ldots, S_{t,i_{F_t}}^{\mathrm{c}}\}$ denote the cache capacity set during time slot t, where S_{t,i_f}^{c} is the cache capacity occupied by the cached file (index $i_f \in \mathcal{C}$). If $S_{i_f} \leq S_{\max,t}$, $S_{t,f}^{\mathrm{c}} = S_{i_f}$, otherwise, $S_{t,f}^{\mathrm{c}} = S_{\max,t}$.

Now, we explain the rationale of setting the maximum size for the cached files $S_{\max,t}$: Firstly, we need to limit the maximum capacity occupied by each cached file to keep the total capacity from exceeding the storage capacity C_{f} of the F-AP when the cached files need to be updated. Secondly, given the limited coverage of the F-AP and user mobility, the F-AP may only need to provide some part of the requested file to the user. For example, it takes time for a user to request and use a large file (such as a video file), and in this process, the user may have left the current F-AP coverage (especially for those users in vehicles), i.e., the F-AP cannot continue to serve the user. Finally, for some files, especially for larger files, users may only need to request part of the content. For example, for large files such as videos, users may only watch the first part of the content, and may stop watching if they find they are not interested in it after all. Besides, $S_{\max,t}$ can be adjusted with the update of cached files to make sure most files can be directly cached instead of being 'pre-split'. The corresponding adjustment mechanism will be described in Part C of Section IV.

In terms of content recommendation, the recommendation policy is incorporated into the cache policy, where the 'abstract' parts (e.g., the thumbnail of an image or the title of an article) of its current cached files are recommended by each F-AP to users within its coverage by periodical broadcasting. This recommendation policy is more suitable for multi-user scenarios. Recommendation can help users find the files they are interested in, thus increasing the number of user requests.

A time-varying user mobility pattern is adopted here. During time slot t, $\mathcal{U}_t = \{1, 2, \ldots, u, \ldots, U_t\}$ denotes the set of the users within coverage of an F-AP in the F-RANs, where $U_t = U_t^{\mathrm{a}} + U_t^{\mathrm{r}}$, U_t^{a} is the number of the newly arrived users in time slot t and users' average arrival number λ follows Poisson distribution, and U_t^{r} is the number of users who are already in coverage of the F-AP during time slot $t-1$ and have remained until the tth time slot. Let $\mathcal{P}_t^{\mathrm{l}} = \{p_{t,1}^{\mathrm{l}}, p_{t,2}^{\mathrm{l}}, \ldots, p_{t,u}^{\mathrm{l}}, \ldots, p_{t,U_t}^{\mathrm{l}}\}$ denote the set of the probabilities of the U_t users leaving coverage of the F-AP. Here, the $\mathcal{P}_t^{\mathrm{l}}$ has a superscript l to distinguish it from the preference vector $\mathcal{P}_t$ in Part A of Section III.

Assume that multiple requests can be made by each user in each time slot. Let $\mathrm{req}_t = \{\mathrm{req}_{t,1}, \mathrm{req}_{t,2}, \ldots, \mathrm{req}_{t,u}, \ldots, \mathrm{req}_{t,U_t}\}$ denote the request set of U_t users during time slot t, where $\mathrm{req}_{t,u} = \{\mathrm{req}_{t,u,1}, \mathrm{req}_{t,u,2}, \ldots, \mathrm{req}_{t,u,n}, \ldots, \mathrm{req}_{t,u,N_{t,u}}\}$ denotes the request set of the uth user during time slot t, $N_{t,u} \in [0, N_{\max}]$ is the number of the requests and $N_{\max}$ is the maximum number of requests per user during time slot t. The request $\mathrm{req}_{t,u,n}$ is expressed as: $\mathrm{req}_{t,u,n} = \langle f_{t,u,n}, t_{u,n}, S_{f_{t,u,n}}^{\mathrm{p}} \rangle$, where $f_{t,u,n} \in \mathcal{C}$ denotes the index of the file which is required by users, $t_{u,n}$ is the time of user requesting, and $S_{f_{t,u,n}}^{\mathrm{p}}$ is the size of the requested part of the corresponding file. Obviously, $S_{f_{t,u,n}}^{\mathrm{p}}$ is less than or equal to the size of the current requested file.

During time slot t, for a request $\mathrm{req}_{t,u,n}$, the corresponding served F-AP needs to check the local caching state of the requested part. That is, the F-AP needs to determine not only whether the requested file with the index $f_{t,u,n}$ is cached, but also whether the requested part is cached by comparing $S_{f_{t,u,n}}^{\mathrm{p}}$ to the current maximum size of the cached files $S_{\max,t}$. Let $\theta_t \in \{0, 1\}$ denote a caching status judging function. If the requested part is cached locally, i.e., the file corresponding to the requested part is cached in the local F-AP and $S_{f_{t,u,n}}^{\mathrm{p}} \leq S_{\max,t}$, $\theta_t(f_{t,u,n}) = 1$ and the part cached in the local F-AP could meet the demand of the uth user. Otherwise, if the F-AP has not cached the file corresponding to the index $f_{t,u,n}$ or the cached part of the file does not meet the demand of the user ($S_{f_{t,u,n}}^{\mathrm{p}} > S_{\max,t}$), $\theta_t(f_{t,u,n}) = 0$. Furthermore, if $\theta_t(f_{t,u,n}) = 1$, the requested file (or the requested part) corresponding to the index $f_{t,u,n}$ can be provided by the local F-AP, which constitutes a 'cache hit'. Otherwise, the requested file needs to be fetched from the cloud server. In the proposed caching architecture, only the case where users fetch files from their closest F-AP or from the cloud server is considered, and the case of multi-F-APs cooperative caching is not considered. The key notations of the paper are summarized in Table I.

TABLE I. Summary of Key Notations

Notation	Definition/Description	Notation	Definition/Description
H	The number of F-APs	t	Discrete time slots, $t = 0, 1, \ldots, T$
$\mathcal{C}$	File index library, $\mathcal{C} = 1, \ldots, c, \ldots, C$	C_{f}	Storage capacity of a single F-AP
S^{c}_{t,i_f}	The actual cache capacity occupied by the cached file (index $i_f \in \mathcal{C}$)	F_t	The number of cached files, $F_t = C_{\mathrm{f}}/S_{\max,t}$
$S_{\max,t}$	The maximum size of the cached files	$S_{\mathrm{av},t}$	The average of the cached files' sizes
$\mathcal{U}_t$	Set of users, $\mathcal{U}_t = \{1, \ldots, u, \ldots, U_t\}$	U_t	The number of users, $U_t = U^{\mathrm{a}}_t + U^{\mathrm{r}}_t$
U^{a}_t	The number of newly arrived users	λ	Average arrival number of Poisson distribution
U^{r}_t	The number of remaining users	$N_{t,u}$	The number of the requests, $N_{t,u} \in [0, N_{\max}]$
$N_{\max}$	The maximum number of requests per user during time slot t	$\mathcal{P}^{\mathrm{l}}_t$	Leaving probabilities of U_t users, $\mathcal{P}^{\mathrm{l}}_t = \left\{p^{\mathrm{l}}_{t,1}, \ldots, p^{\mathrm{l}}_{t,u}, \ldots, p^{\mathrm{l}}_{t,U_t}\right\}$
req_t	The request set of users, $\mathrm{req}_t = \{\mathrm{req}_{t,1}, \ldots, \mathrm{req}_{t,u}, \ldots, \mathrm{req}_{t,U_t}\}$	$\mathrm{req}_{t,u}$	The request set of the uth user, $\mathrm{req}_{t,u} = \{\mathrm{req}_{t,u,1}, \ldots, \mathrm{req}_{t,u,N_{t,u}}\}$
$\mathrm{req}_{t,u,n}$	The nth request of the uth user, $\mathrm{req}_{t,u,n} = \langle f_{t,u,n}, t_{u,n}, S^{\mathrm{p}}_{f_{t,u,n}}\rangle$	θ_t	A caching status judging function, $\theta_t \in \{0, 1\}$
$f_{t,u,n}$	index of the requested file, $f_{t,u,n} \in \mathcal{C}$	$t_{u,n}$	accurate requesting time
$S^{\mathrm{p}}_{f_{t,u,n}}$	The size of the requested part	π	The caching policy
$\mathcal{P}_t$	The preference vector set of U_t users, $\mathcal{P}_t = \{\boldsymbol{p}_{t,1}, \ldots, \boldsymbol{p}_{t,u}, \ldots, \boldsymbol{p}_{t,U_t}\}$	$\boldsymbol{p}_{t,u}$	A preference vector, $\boldsymbol{p}_{t,u} = [p_{t,u,1}, \ldots, p_{t,u,c}, \ldots, p_{t,u,C}]^T$
$p_{t,u,f_{t,u,n}}$	The preference value of the file (index $f_{t,u,n}$)	$\mathcal{P}^0$	A candidate set, $\mathcal{P}^0 = \{\boldsymbol{p}^0_1, \ldots, \boldsymbol{p}^0_g, \ldots, \boldsymbol{p}^0_G\}$
$\boldsymbol{p}^0_g$	An initial preference vector, $\boldsymbol{p}^0_g = [p^0_{g,1}, \ldots, p^0_{g,c}, \ldots, p^0_{g,C}]^T$	$\mathcal{AC}$	{Inactiveness} parameters of U_t users, $\mathcal{AC} = \{ac_1, \ldots, ac_u, \ldots, ac_{U_t}\}$
ac_u	{Inactiveness} parameter of the uth user, $ac_u \in \{0, 1\}$	$\mathbf{rec}_t$	Recommendation status, $\mathbf{rec}_t = [\mathrm{rec}_{t,1}, \ldots, \mathrm{rec}_{t,c}, \ldots, \mathrm{rec}_{t,C}]^T$
β	Recommendation factor, $\beta \geq 1$	μ	A small user preference value
$\mathcal{P}^{\mathrm{i}}_t$	Impulsive requesting probabilities of users, $\mathcal{P}^{\mathrm{i}}_t = \{\mathcal{P}^{\mathrm{i}}_{t,1}, \ldots, \mathcal{P}^{\mathrm{i}}_{t,u}, \ldots, \mathcal{P}^{\mathrm{i}}_{t,U_t}\}$	$p^{\mathrm{i}}_{t,u}$	Impulsive request probabilities of the uth user $\mathcal{P}^{\mathrm{i}}_{t,u} = \{p^{\mathrm{i}}_{t,u,1}, \ldots, p^{\mathrm{i}}_{t,u,n}, \ldots, p^{\mathrm{i}}_{t,u,N_{t,u}}\}$
$p^{\mathrm{i}}_{t,u,n}$	Impulsive request probability	p_{fre}	A parameter adjusting user request frequency
b	Unit transmission cost ($b \gg s$) (from cloud server to users)	η	A scaling parameter (MNOs charges for each unit file)
s	Unit transmission cost (from F-AP to users)	$c(t)$	The updating transmission cost
Ω	The long-term net profit	γ	The discount factor of RL, $\gamma \in (0, 1]$
ϵ	Greedy factor of ϵ-greedy method, $\epsilon \in (0, 1)$	D	The replay {memory} with the capacity B
ξ	The threshold of cache hit rate of the F-AP	$\mathcal{M}$	The set of some requested files that cannot be obtained from the F-AP

III. Proposed Time-Varying Personalized User Request Model

In this section, considering user behavior, user mobility pattern, user preferences affected by content recommendation and some exceptive circumstances in reality, we propose a time-varying personalized user request model to characterize the dynamic user demands. Note here that the knowledge of this user request model is unknown in advance for the subsequent caching optimization.

A. User Preference and Influence Factors

Consider the personalized and time-varying characteristics of the user preference. Let $\mathcal{P}_t = \{\boldsymbol{p}_{t,1}, \ldots, \boldsymbol{p}_{t,u}, \ldots, \boldsymbol{p}_{t,U_t}\}$ denote the preference vectors of the U_t users during time slot t, where $\boldsymbol{p}_{t,u} = [p_{t,u,1}, p_{t,u,2}, \ldots, p_{t,u,C}]^T$ represents the preference vector with dimension C describing the preference of the uth user, i.e., the probability distribution of the uth user requesting each file in the file library $\mathcal{C}$ and the sum of all the elements in the preference vector $\boldsymbol{p}_{t,u}$ equals 1. The Mandelbrot-Zipf (M-Zipf) distribution is usually utilized to characterize the content popularity distribution [41], and to match this content popularity distribution, the user preference vector should also conform to the M-Zipf distribution. In addition, different users may have similar preferences, i.e., preference vectors of different users may overlap to some extent. Let $\mathcal{P}^0 = \{\boldsymbol{p}^0_1, \boldsymbol{p}^0_2, \ldots, \boldsymbol{p}^0_g, \ldots, \boldsymbol{p}^0_G\}$, with a superscript 0 to distinguish it, denote a candidate set containing G initial user preference vectors, where $\boldsymbol{p}^0_g = [p^0_{g,1}, p^0_{g,2}, \ldots, p^0_{g,C}]^T$ represents an initial user preference vector following the M-Zipf distribution as follows:

$$p_{g,c} = \frac{(i_c + \tau)^{-\alpha}}{\sum_{j \in \mathcal{C}} (i_j + \tau)^{-\alpha}}, \forall c \in \mathcal{C}, \tag{1}$$

where i_c represents the rank of content c in descending order of global content popularity, α is the skewness factor and τ is plateau factor. For a user who has newly arrived, its preference vector is an initial user preference vector randomly selected from $\mathcal{P}^0$.

The schematic of the formation and influence process of the user preference vector is shown in Fig. 2, where the candidate set $\mathcal{P}^0$ has $G = 4$ initial user preference vectors and the value of the dimension C of each initial user preference vector is 6. For convenience of representation and subsequent calculation, the corresponding preference values of the preference vectors are all reference values and may not exactly fit the Zipf distribution. As illustrated in Fig. 2, the formation process applies only to the newly arrived users. When a user arrives in the F-AP coverage, the formation process of the corresponding

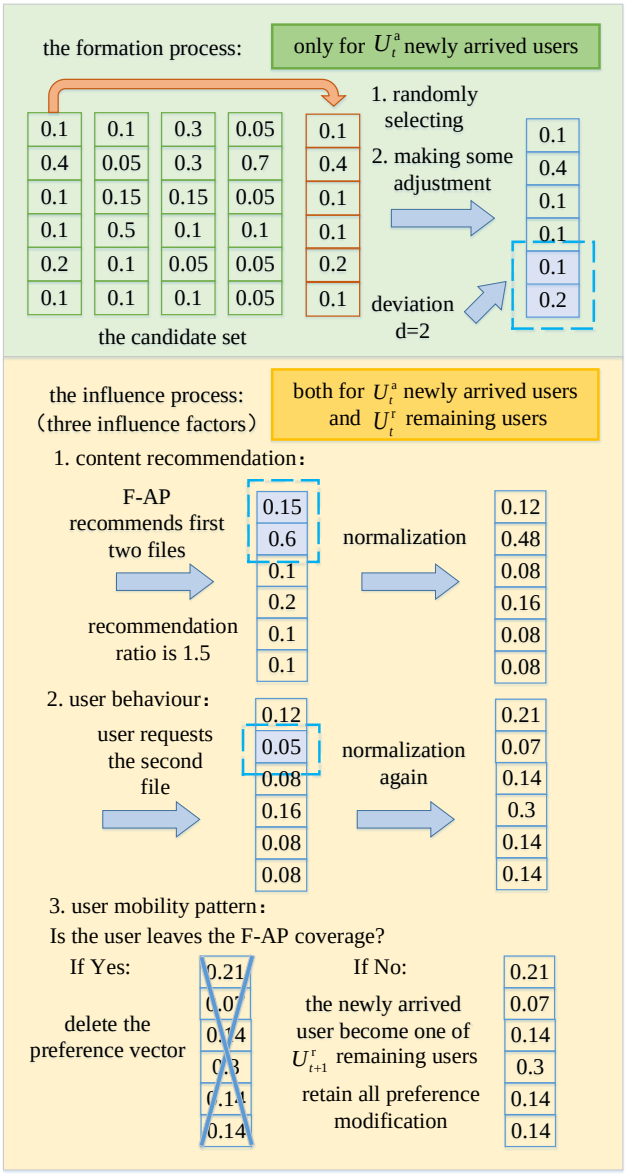


Fig. 2. The schematic of the formation and influence process of the user preference vector.

preference vector can be divided into two steps. Firstly, an initial user preference vector is randomly selected from the candidate set $\mathcal{P}^0$. Of course, the random selection is not necessarily the best choice. For the objectivity of modeling and simulation, we choose random selection here. In Fig. 2, the first initial user preference vector is selected from the candidate set $\mathcal{P}^0$. Secondly, some adjustment is made on the selected vector. As shown in Fig. 2, the last two values of the preference vector are switched.

In addition, according to [42], the request frequencies of users are often heterogeneous, that is, less than 20% of users (active users) generate 80% of daily network traffic, i.e., the majority of user requests are generated by those most active users. Therefore, we now introduce the inactiveness parameter collection $\mathcal{AC} = \{ac_1, ac_2, \ldots, ac_u, \ldots, ac_{U_t}\}$ to represent the active levels of U_t users in the F-AP coverage, where $ac_u \in \{0, 1\}$, $ac_u = 0$ indicates that the uth user is an active user who makes frequent file requests and $ac_u = 1$ indicates the opposite case. This inactiveness parameter ac_u is set as a fixed parameter of each user in this paper (it can also be extended to a time-varying parameter). Therefore, we only need to set the values of this inactiveness parameters for the U_t^a newly arrived users at the beginning of current time slot and the values of inactiveness parameters of the U_t^r users have already been set in some previous time slot. To match the heterogeneity law of user request frequency, for each newly arrived user, the value of its inactiveness parameter only has probability of 20% that it is equal to 0, i.e., only 20% of newly arrived users are active users.

Considering that user preference vectors may be affected by some external factors, we take $\boldsymbol{p}_{t,u}$ as an example to describe their influence patterns. As shown in Fig. 2, in the evolution process, there are three influence factors that may affect user preference vectors. Note that the influence process applies to the preference vectors of all users (U_t^a newly arrived users and U_t^r remaining users) in the current F-AP coverage. In Fig. 2, the preference vector of a newly arrived user in the formation process is used as an example to illustrate the influence process.

(1) ***Content recommendation:*** According to the recommendation policy, the 'abstract' parts (the thumbnail of an image or the title of an article) of its current cached files are recommended by the F-AP to the users within its service coverage at the beginning of each time slot. When the cached files are refreshed, the recommended contents are also modified. The impact of recommendation is quantized by recommendation factor β ($\beta \geq 1$) . The recommendation status of the F-AP is denoted by $\mathbf{rec}_t = [\mathrm{rec}_{t,1}, \mathrm{rec}_{t,2}, \ldots, \mathrm{rec}_{t,c}, \ldots, \mathrm{rec}_{t,C}]^T$ in time slot t. For $\mathrm{rec}_{t,c}, \forall c \in \mathcal{C}$, if the 'abstract' part of the cth file is recommended, $\mathrm{rec}_{t,c} = \beta$, and $\mathrm{rec}_{t,c} = 1$ otherwise. It is assumed that content recommendation influences the user preference with an 'interacting' pattern, which can be expressed as follows:

$$\boldsymbol{p}_{t,u} \leftarrow \Phi(\boldsymbol{p}_{t,u} \circ \mathbf{rec}_t), \tag{2}$$

where $\circ$ denotes the operator of Hadamard product and Φ is a normalization function to ensure that all elements of $\boldsymbol{p}_{t,u} \circ \mathbf{rec}_t$ still add up to 1. This 'interacting' mode reflects the mutual influence between content recommendation and user preferences, i.e., a better recommendation effect can be obtained by recommending content with larger user preference value to users. We assume that the content preference vector of users will be immediately updated once the recommended content is broadcast.

For the content recommendation influence factor, as shown in Fig. 2, the F-AP recommends the first two files in the file library $\mathcal{C}$. The preference values (0.1 and 0.4) of the corresponding position in the current preference vector are multiplied by the recommendation ratio (1.5) becoming the new preference values (0.15 and 0.6). Then, the modified user preference vector is normalized to ensure that all elements of the vector still add up to 1.

(2) ***User behaviour:*** Note that users barely make repeated

requests for the same file. During time slot t, for the nth request $\mathrm{req}_{t,u,n} = \langle f_{t,u,n}, t_{u,n}, S^{\mathrm{p}}_{f_{t,u,n}} \rangle$ of the uth user, the preference value of the requested file is set to a small value μ, i.e., $p_{t,u,f_{t,u,n}} = \mu$, and the modified user preference is normalized to ensure all its elements still add up to 1. Besides, the above preference modification is also affected by the order of requests (order by $t_{u,n}$). That is, the modification caused by the nth request $\mathrm{req}_{t,u,n}$ needs to be made according to the user preference vector modified by the $(n-1)$th request $\mathrm{req}_{t,u,n-1}$, and it can be expressed by a function A_n:

$$A_n(A_{n-1}, \mathrm{req}_{t,u,n}, \mu) \rightarrow \boldsymbol{p}_{t,u}, \tag{3}$$

where A_{n-1} represents the user preference vector modified by the previous $(n-1)$ requests. During time slot t, $N_{t,u}$ requests $\mathrm{req}_{t,u} = \{\mathrm{req}_{t,u,1}, \mathrm{req}_{t,u,2}, \ldots, \mathrm{req}_{t,u,n} \ldots, \mathrm{req}_{t,u,N_{t,u}}\}$ will in turn affect $\boldsymbol{p}_{t,u}$, and it can be expressed as follows:

$$\begin{aligned} \boldsymbol{p}_{t,u} \leftarrow & A_{N_{t,u}} = A_{N_{t,u}}\Big(A_{N_{t,u}-1}\big(\ldots A_n(\ldots A_2(A_1 \\ & (\boldsymbol{p}_{t,u}, \mathrm{req}_{t,u,1}, \mu), \mathrm{req}_{t,u,2}, \mu), \ldots, \mathrm{req}_{t,u,n}, \mu), \\ & \ldots, \mathrm{req}_{t,u,N_{t,u}-1}, \mu\big), \mathrm{req}_{t,u,N_{t,u}}, \mu\Big), \end{aligned} \tag{4}$$

where $N_{t,u}$ requests correspond to $N_{t,u}$ modifications and each modification should be made immediately after each request. Otherwise, the same file can be requested repeatedly.

As illustrated in Fig. 2, the user only requests one file, i.e., the second file in the file index library $\mathcal{C}$, during the current time slot. The preference value (0.48) of the corresponding position in the current preference vector is set to a small value $\mu = 0.05$. Then, the modified user preference vector is normalized again to ensure that all elements of the vector still add up to 1.

(3) ***User mobility pattern:*** Note that the influence of recommendation can be delayed, and the user preference modifications caused by avoiding repeated requests are continued before the user leaves coverage of the current F-AP. All their preference modifications are retained before they leave coverage of the current F-AP for U_t^{r} remaining users,. Once the user leaves the F-AP coverage, the corresponding preference vector will be deleted (the user will not make any file requests within the current F-AP coverage). This is also the largest difference between the preferences of U_t^{r} remaining users and U_t^{a} newly arrived users. The former is 'inherited' and determined, whereas the latter is randomly selected from the candidate set $\mathcal{P}^0$ with different adjustments.

As illustrated in Fig. 2, if the user leaves coverage of the F-AP (both U_t^{a} newly arrived users and U_t^{r} remaining users), the corresponding preference vector will be deleted. Otherwise, the user will be one of the U_{t+1}^{r} remaining users of the next time slot $t+1$ and all preference modifications will be retained. And the next round of the influence process will start in the next time slot.

B. User Request Model

Our proposed time-varying personalized user request model includes two parts: the user impulsive request and the user preference based request, which is illustrated in Fig. 3. In either case, once a request $\mathrm{req}_{t,u,n}$ occurs, it is a user behaviour factor which will influence the current user preference $\boldsymbol{p}_{t,u}$.

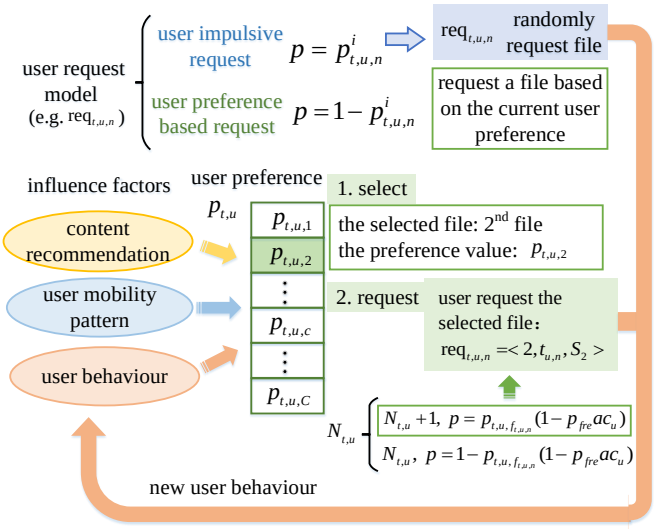


Fig. 3. The schematic of the proposed user request model.

(1) ***User impulsive request:*** Due to some special cases in reality, a user may impulsively request a file or the user has to request a particular file for some reason, and most of these requests are not in accordance with their user preferences. In this case, the common situation is that when a file is needed, the file must be requested. Let $\mathcal{P}_t^{\mathrm{i}} = \{\mathcal{P}_{t,1}^{\mathrm{i}}, \mathcal{P}_{t,2}^{\mathrm{i}}, \ldots, \mathcal{P}_{t,u}^{\mathrm{i}}, \ldots, \mathcal{P}_{t,U_t}^{\mathrm{i}}\}$, with a superscript i to distinguish it, denote the impulsive requesting probability set of U_t users during time slot t, where $\mathcal{P}_{t,u}^{\mathrm{i}} = \{p_{t,u,1}^{\mathrm{i}}, p_{t,u,2}^{\mathrm{i}}, \ldots, p_{t,u,n}^{\mathrm{i}}, \ldots, p_{t,u,N_{t,u}}^{\mathrm{i}}\}$ denotes the impulsive requesting probabilities of the uth user for its $N_{t,u}$ requests and $N_{t,u} \in [0, N_{\max}]$. That is, for a request $\mathrm{req}_{t,u,n}$, the probability of impulsive requesting is $p_{t,u,n}^{\mathrm{i}}$, and the probability of requesting a file according to the user preference is $1 - p_{t,u,n}^{\mathrm{i}}$. In addition, by considering that the impulsive request is often involuntary or directed, this type of user requests has little to do with the inactiveness parameter collection $\mathcal{AC}$ of users. Therefore, we assume that the status of user requests (or the frequency of user requests) in this case is independent of the inactiveness parameters.

(2) ***User preference based request:*** The user preference based request, with probability $1 - p_{t,u,n}^{\mathrm{i}}$, is divided into two steps: selecting the file and requesting the file. Next, we take preference $\boldsymbol{p}_{t,u}$ of the uth user as an illustration to elaborate the forming process of its nth request $\mathrm{req}_{t,u,n}$ in time slot t. The schematic is shown in Fig. 3.

Firstly, the user randomly selects a file according to the current user preference $\boldsymbol{p}_{t,u}$, i.e., the probability of a file being selected is proportional to its preference value. In Fig. 3, $f_{t,u,n} = 2$, i.e., the 2nd file in $\mathcal{C}$ is selected and its preference value is $p_{t,u,f_{t,u,n}} = p_{t,u,2}$ (for description convenience, the selected file is also represented by $f_{t,u,n}$). Secondly, the user requests the selected file. The Bernoulli distribution is utilized to model the request process, which determines whether the selected file will be requested or not, and it can be expressed

as follows:

$$N_{t+1,u} = \begin{cases} N_{t,u}+1, & p = p_{t,u,f_{t,u,n}}(1-p_{\mathrm{fre}}ac_u), \\ N_{t,u}, & p = 1-p_{t,u,f_{t,u,n}}(1-p_{\mathrm{fre}}ac_u), \end{cases} \tag{5}$$

where p_{fre} is a constant parameter used to adjust the user request frequency in line with the user inactiveness parameter ac_u. ac_u is a binary value to indicate whether a user is an active user or not. When ac_u = 0, which indicates the user is active, $p_{\mathrm{fre}}ac_u = 0$ and p_{fre} loses its regulating effect. For the uth user, $p = p_{t,u,f_{t,u,n}}(1-p_{\mathrm{fre}}ac_u)$ is the probability of requesting the selected file, so the probability of not requesting the selected file is $p = 1-p_{t,u,f_{t,u,n}}(1-p_{\mathrm{fre}}ac_u)$. Obviously, the user preference based request is related to p_{fre} and the inactiveness parameter ac_u. As shown in Fig. 3, the selected file is requested and $\mathrm{req}_{t,u,n} = \langle 2, t_{u,n}, S_2\rangle$, which will influence $\boldsymbol{p}_{t,u}$ as the nth 'user behaviour' influence factor. Note that once $N_{t,u} = N_{\max}$, the user no longer selects and requests any file.

When $ac_u = 0$, i.e., the user is an active user, and the formula (5) can be expressed as:

$$N_{t+1,u} = \begin{cases} N_{t,u}+1, & p = p_{t,u,f_{t,u,n}}, \\ N_{t,u}, & p = 1-p_{t,u,f_{t,u,n}}, \end{cases} \tag{6}$$

where the user request is not adjusted by the parameter p_{fre}. On the contrary, when $ac_u = 1$, i.e., the user is not an active user, and the formula (5) can be expressed as:

$$N_{t+1,u} = \begin{cases} N_{t,u}+1, & p = p_{t,u,f_{t,u,n}}(1-p_{\mathrm{fre}}), \\ N_{t,u}, & p = 1-p_{t,u,f_{t,u,n}}(1-p_{\mathrm{fre}}), \end{cases} \tag{7}$$

where the user request is adjusted by the parameter p_{fre}. In (6) and (7), the larger the user preference value of a file, the more likely it is to be selected and requested.

Given that majority of user requests (e.g. 80%) are generated by active users (e.g. 20%), we control the user request distribution by setting a reasonable value for parameter p_{fre}. A reasonable value of p_{fre} should satisfy the following condition:

$$\frac{20\times 1}{20\times 1 + 80\times p_{\mathrm{fre}}} = 0.8. \tag{8}$$

Obviously, the value of p_{fre} that can satisfy this condition is approximately 0.06.

Based on the proposed user request model, during time slot t, when the uth user attempts to make the nth request $\mathrm{req}_{t,u,n}$, the request situation can be divided into two cases. The first case is that the request $\mathrm{req}_{t,u,n}$ is an impulsive request with the probability $p^{\mathrm{i}}_{t,u,n}$, and the corresponding file will be requested directly. The second case is that the nth request $\mathrm{req}_{t,u,n}$ is a user preference based request with the probability $1-p^{\mathrm{i}}_{t,u,n}$, where the generation of a user request involves two steps (select the file and execute the request). Note that the selected file may not be finally requested in the second case, i.e., the nth user request $\mathrm{req}_{t,u,n}$ may not actually be made in the second case. Of course, when this situation occurs, for the uth user, the identifier of the next request during time slot t is still $\mathrm{req}_{t,u,n}$, i.e., $\mathrm{req}_{t,u,n}$ is the user request that is successfully made.

For each request $\mathrm{req}_{t,u,n}$, the value of its parameter $S^{\mathrm{p}}_{f_{t,u,n}}$ is randomly generated, which is no larger than the size of the current requested file. Once $N_{t,u} = N_{\max}$, the uth user no longer requests any files in either case. As illustrated in Fig. 3, in either case, once the request $\mathrm{req}_{t,u,n}$ occurs, the preference vector $\boldsymbol{p}_{t,u}$ of the uth user will be affected as the nth 'user behaviour' influence factor.

IV. Proposed DDQN-Based Distributed Edge Caching Algorithm with Content Recommendation

A. Problem Formulation

The objective of our paper is to find a caching policy π (to consistent with the subsequent RL, π is used here) with content recommendation by maximizing the long-term net profit of each F-AP.

Take the nth request $\mathrm{req}_{t,u,n} = \langle f_{t,u,n}, t_{u,n}, S^{\mathrm{p}}_{f_{t,u,n}}\rangle$ of the uth user as an example. The F-AP directly transmits the cached file with the index $f_{t,u,n}$ to the user if the caching status $\theta_t(f_{t,u,n}) = 1$, and the transmission consumption is denoted by $s \times S^{\mathrm{p}}_{f_{t,u,n}}$, where s is the transmission cost required to transmit a file with a unit size from the F-AP to the user and $S^{\mathrm{p}}_{f_{t,u,n}}$ is the size of the requested content. Otherwise, $\theta_t(f_{t,u,n}) = 0$, the consumption for the cloud server transmitting the requested file to the user is denoted by $b \times S^{\mathrm{p}}_{f_{t,u,n}}$ ($b \gg s$), where b is the transmission cost required to transmit a file with a unit size from the cloud server to the user. Since the transmission distance from the cloud server to the user is far greater than the distance from the F-AP to the user, $b \gg s$.

Moreover, $(b-s)\times S_{f_{t,u,n}}$ denotes the cost for transmitting a file from the cloud server to the F-AP for caching update. The reason why $S_{f_{t,u,n}}$ is used here instead of $S^0_{t,f_{t,u,n}}$ is that the entire updated file needs to be fetched before determining whether the file needs to be split during the update process, where $S^0_{t,f_{t,u,n}}$ is the actual cache capacity occupied by the cached file with the index $f_{t,u,n}$ in the F-AP during time slot t. In each time slot, the caching updates of the F-AP may happen. The cost of the F-AP for files updating is set to $c(t)$ in time slot t, which is a multiple of $(b-s)\times S_{f_{t,u,n}}$. In addition, the corresponding transmission cost could be ignored since only the F-AP will recommend the 'abstract' parts from the current cached files to users. Correspondingly, in time slot t, the total transmission cost is written as follows:

$$\sum_{u=1}^{U_t}\sum_{n=1}^{N_{t,u}} \Big\{\theta_t(f_{t,u,n})\times s \times S^{\mathrm{p}}_{f_{t,u,n}} + [1-\theta_t(f_{t,u,n})] \times (b \times S^{\mathrm{p}}_{f_{t,u,n}})\Big\} + c(t). \tag{9}$$

Then, the long-term net profit Ω of the F-AP is formulated as follows:

$$\Omega = \sum_{t=0}^{T}\Big\{\sum_{u=1}^{U_t}\sum_{n=1}^{N_{t,u}} \big[\eta \times S^{\mathrm{p}}_{f_{t,u,n}} - \theta_t(f_{t,u,n})\times s \times S^{\mathrm{p}}_{f_{t,u,n}} - (1-\theta_t(f_{t,u,n}))\times (b\times S^{\mathrm{p}}_{f_{t,u,n}})\big] - c(t)\Big\}, \tag{10}$$

where η denotes a scaling parameter indicating how much the MNOs charge a user for each unit file [21], [30]. Therefore, we

can express the edge caching optimization problem as follows:

$$\max_{\pi} \Omega, \tag{11}$$

$$\text{s.t. } 0 \leq \frac{c(t)}{(b-s) \times S_{f_{t,u,n}}} \leq F_t, \tag{11a}$$

$$0 \leq N_{t,u} \leq N_{\max}, \tag{11b}$$

where (11a) denotes the cache capacity constraint of the F-AP for the updating process of the cached files and (11b) denotes the constraint for the number of requests. The caching policy π here is used to determine which files need to be stored in the current F-AP.

B. RL Framework

For problem (11), it is hard to acquire the optimal theoretical solution by using traditional optimization methods. By considering the personalized and time-varying characteristics of the proposed user request model, RL framework in machine learning is utilized next to solve the problem.

First, the objective in problem (11) is the maximum of net profit, which is the long term reward. Correspondingly, the problem (11) is modeled as a Markov decision process (MDP), which is the basic model of RL. Then in each time slot t, the problem (11) can be solved with an RL framework, where the *agent* continuously interacts with and learns from the *environment* [43], which constitutes an *episode*. The agent can select an *action* $\boldsymbol{a}(t)$ in the current *environment state* $\boldsymbol{s}(t)$ during each time slot t. The environment feeds back a *reward* $r(t)$ to the *agent* after this *action* $\boldsymbol{a}(t)$ is executed which can let the environment go into a next new *state* $\boldsymbol{s}(t+1)$. The state transition probability can be expressed as follows:

$$P(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a}) = P[\boldsymbol{s}(t+1) = \boldsymbol{s}'|\boldsymbol{s}(t) = \boldsymbol{s}, \boldsymbol{a}(t) = \boldsymbol{a}], \tag{12}$$

where $\sum_{\boldsymbol{s}' \in \mathcal{S}} P(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a}) = 1$.

Let $\boldsymbol{e}(t) = [\boldsymbol{s}(t), \boldsymbol{a}(t), r(t), \boldsymbol{s}(t+1)]^T$ denote an *experience* vector of the above interaction. In the current environment state, the *policy* π is exploited to decide which actions will be selected for execution. According to the reward function, RL can guide the agent to select experience vectors from the experience pool, and optimize our caching strategy by maximizing long-term reward expectations.

We model the problem (11) as an MDP in an F-AP. The external environment of RL is the dynamic user request, which has been constructed in part B of section III, and the learning agent is the F-AP. The state space, action space and reward function are described as follows.

(1) ***State Space*** $\mathcal{S}$*:* The state space $\mathcal{S}$ mainly includes the content cache state in the F-AP and content request state from users. The request state from users has been modeled by $\mathrm{req}_{t,u,n}$ and the cache state is $s(t)$. The cache state $s(t)$ is the index collection of the files currently cached in the F-AP, i.e., $s(t) = [i_1, i_2, \ldots, i_f, \ldots, i_{F_t}]^T,\ \forall i_f \in [1, C]$. However, the types of states rapidly increase with C, which may lead to the curse of dimensionality in RL. Therefore, we assume that the index collection is based on reverse ordering of the cumulative number of requests, where some infrequent states will rarely appear, which is enlightened by the least frequently used (LFU) method [44].

The initial value of $\boldsymbol{s}(t)$ is determined by the initial cached files of the F-AP. Obviously, a direct and convenient way is to randomly select F_t files from the cloud server as the initial cached files in the F-AP. In [45] and [46], the authors proposed a transfer learning method to improve popularity profile estimation by using prior knowledge of information provided from proxy sources, e.g., social networks. Similarly, we can set the initial cached files in the F-AP by referring to the popular files from an alternative domain with the transfer learning approach, which is more efficient than the random approach and can increase the rate of subsequent initial training process. However, the effectiveness of this approach depends on how well the proxy source, e.g., Facebook, represents user requests for the target domain. When file popularity varies greatly between the target domain and the proxy source, the caching decision can be misguided, while excess processing overhead is imposed to the F-AP or the network operator. In addition, when considering collaborative caching, the initial cached files for the current F-AP can be determined by referring to the nearby F-AP that is in the training process or has already completed training.

(2) ***Action Space*** $\mathcal{A}$: The action space $\mathcal{A}$ indicates whether the cached files in the F-AP are to be updated during time slot t, i.e., $\boldsymbol{a}(t) \in \{0, 1\}$. Based on req_t, the current cached files in the F-AP are reordered and the request frequency is updated. Let parameter ξ denote the threshold of cache hit rate, which can be adjusted during the training process. Let a binary variable $m(t)$ denote the cache miss condition during time slot t. If the cache hit rate of the F-AP cannot reach ξ, $m(t) = 1$, some requested files (which may be from different users) cannot be fetched from the F-AP, and these files construct a set $\mathcal{M}(\mathcal{M} \neq \varnothing)$. And we restrict $\mathcal{M}$ so that it does not contain the same files. Otherwise, $m(t) = 0$ and $\mathcal{M} = \varnothing$. Correspondingly, the next state $\boldsymbol{s}(t+1)$ is determined by $\boldsymbol{a}(t)$ and $m(t)$ jointly. If $m(t) = 0$ and $\boldsymbol{a}(t) = 1$, or $\boldsymbol{a}(t) = 0$, $\boldsymbol{s}(t+1)$ denotes the index collection of the newly sorted cached files. Otherwise, if $m(t) = 1$ and $\boldsymbol{a}(t) = 1$, the F-AP replaces the cached file currently located at the end of the storage space by a file randomly chosen from $\mathcal{M}$, and $\boldsymbol{s}(t+1)$ denotes the index collection after the operation of sorting and replacement. The updating policy can update at most one file in one time slot, which reduces the inefficient update caused by unusual user requests and satisfies the constraint of cache capacity in $(11a)$. Besides, in order not to affect the training process of the next time slot, the previous set $\mathcal{M}$ is emptied at the beginning of each time slot.

(3) ***Reward Function:*** By considering the objective function in (11), the reward function is designed as follows:

$$\begin{aligned} r(t) = \sum_{u=1}^{U_t} \sum_{n=1}^{N_{t,u}} \Big\{ & \eta \times S_{f_{t,u,n}} - \theta_t(f_{t,u,n}) \times s \times S^0_{t,f_{t,u,n}} - \\ & [1 - \theta_t(f_{t,u,n})] \times (b \times S_{f_{t,u,n}}) \Big\} - \boldsymbol{a}(t) \times m(t) \times \\ & (b-s) \times S_{f_{t,u,n}}. \end{aligned} \tag{13}$$

Moreover, the action-value function $Q_\pi(\boldsymbol{s}, \boldsymbol{a})$ following the

policy π is defined as follows [43]:

$$\begin{aligned} Q_\pi(\boldsymbol{s},\boldsymbol{a}) &= \mathbb{E}\Big[\sum_{\tau=t}^{T}\gamma^{\tau-t}r(\tau)\Big]_{\boldsymbol{s}(\tau)=\boldsymbol{s},\boldsymbol{a}(\tau)=\boldsymbol{a}.} \\ &= \mathbb{E}\Big[r(t)+\gamma Q_\pi(\boldsymbol{s}',\boldsymbol{a}')\Big]_{\boldsymbol{s}(\tau)=\boldsymbol{s},\boldsymbol{a}(\tau)=\boldsymbol{a}.} \end{aligned} \tag{14}$$

where $\boldsymbol{s}'$ and $\boldsymbol{a}'$ represent $\boldsymbol{s}(t+1)$ and $\boldsymbol{a}(t+1)$, respectively. The discount factor $\gamma \in (0,1]$ implies the current strategy π taking into account the impact of future rewards. Solving reinforcement learning is equivalent to optimizing the Bellman expectation equation for the second equality.

Let $Q^*(\boldsymbol{s},\boldsymbol{a}) = \max_\pi Q_\pi(\boldsymbol{s},\boldsymbol{a})$ denote the optimal action-value function. Correspondingly, the optimal policy π^* can be obtained as follows [43]:

$$\pi^* = \arg\max_{\boldsymbol{a}} Q^*(\boldsymbol{s},\boldsymbol{a}). \tag{15}$$

C. 'Pre-Split' Operation and 'Lazy' Update Process

According to the introduction of the 'pre-split' operation in the system model, when the size of the file to be cached is greater than $S_{\max,t}$ during time slot t, the F-AP needs to split the file and only cache the corresponding content with the size of $S_{\max,t}$. Obviously, this 'pre-split' operation imposes additional processing cost. To control the frequency of this 'pre-split' operation, we need to build a connection between $S_{\max,t}$ and the cached files in the storage space of the F-AP. In time slot t, let $S_{\text{av},t}$ denote the average size of current cached files in the F-AP, i.e.,

$$S_{\text{av},t} = \frac{\sum_{f_{t,u,n}=i_1}^{i_{F_t}} S_{f_{t,u,n}}}{F_t}. \tag{16}$$

Set the value of $S_{\max,t}$ to a multiple (for example, 2 times) of $S_{\text{av},t}$. At this point, the relationship between $S_{\max,t}$ and current cached files in the F-AP is established with the parameter $S_{\text{av},t}$.

Ideally, the value of $S_{\max,t}$ should be updated dynamically in real-time while the cached files are also being updated in the local F-AP. However, the real-time dynamic update of $S_{\max,t}$ will cause several problems. It can lead to the confusion in updating the cached files. Based on the definition of F_t, i.e., $F_t = C_{\text{f}}/S_{\max,t}$, the storage capacity C_{f} of the F-AP is a fixed value and the real-time update of $S_{\max,t}$ will directly lead to the real-time update of F_t. Each update of $S_{\max,t}$ and F_t will lead to a large number of cached files being updated in the F-AP, in which the corresponding cached files are required to be fetched from the cloud server again and undergo the 'pre-split' operation with the updated $S_{\max,t}$, thus resulting in large transmission costs. On the other hand, the value of F_t directly corresponds to the dimension of the state ($\boldsymbol{s}(t) = [i_1, i_2, \ldots, i_f, \ldots, i_{F_t}]^T$). The real-time update of $S_{\max,t}$ and F_t results in the real-time update of the dimension of the state vector ($\boldsymbol{s}(t)$). However, during a period of training process, the agent needs to ensure that the dimension of state $\boldsymbol{s}(t)$ remains constant. The reason is that the state transition with variable dimensions (for example, the state $\boldsymbol{s}(t_1)$ with dimension 10 changes to the state $\boldsymbol{s}(t_2)$ with dimension 5) will bring great challenges to the training process of reinforcement learning.

According to the above analysis, we take a 'lazy' approach to update $S_{\text{av},t}$, $S_{\max,t}$ and F_t. We set the update period of $S_{\text{av},t}$, $S_{\max,t}$ and F_t as one training period, such as 2000 time slots, which can be adjusted according to the training situation.

The 'lazy' scheme means that the update process occurs only at the last time slot $t = T$ for each training period. This period is defined by default but it can also be dynamically modified in the training process. An entire 'lazy' update process includes the following steps: (1) Obtain the file sizes of current cached files in the F-AP and update the value of $S_{\text{av},t}$ according to formula (16), and determine the updated $S_{\max,t}$ and F_t in turn according to the updated $S_{\text{av},t}$. (2) Update all cached files in the F-AP. The update operation here is different from the update operation of $\boldsymbol{a}(t) = 1$ where at most one cached file is updated. Firstly, all cached files of the current F-AP are deleted. Then, F_t files (updated F_t) are fetched from the cloud server based on the index order of the current state $\boldsymbol{s}(T)$ and are cached in the F-AP after the 'pre-split' operation with the updated $S_{\max,t}$. For this update process, if the value of the updated F_t is less than or equal to its original value, we only need to carry out F_t file fetching and 'pre-split' operations successively. Otherwise, additional files are required to be fetched from the cloud server to meet the updated value of F_t. Finally, set the cumulative number of requests for all cached files to 0. (3) The state space $\mathcal{S}$ is cleared, and the elements of initial state $\boldsymbol{s}(0)$ for the next training period are indexes of the files currently cached in the F-AP with the updated F_t as the dimension. Besides, when the RL or DRL framework utilized involves the replay memory, the replay memory also needs to be cleared. At this point, the 'lazy' update process ends, and the training process enters the next training period.

Cache fragmentation issues are inevitable in every cache scenario. The proposed 'pre-split' procedure with a dynamic upper limit can dynamically find a better storage upper limit size, so as to minimize the waste of cache resources caused by storage fragmentation. Although the 'lazy' update process still result in all cached files of the F-AP being updated, the frequency of its occurrences is not high (e.g., one training period includes 2000 time slots), and the total relevant transmission costs of a 'lazy' update process can be considered as a fixed value. As for whether the size limit of each cached content will cause additional cache update costs, this situation generally occurs when the current popular files are all large files and the corresponding popular contents are frequently changed. This scenario occurs infrequently, and additional caching costs can be minimized by limiting the frequency of changes to the dynamic file cache upper limit. Therefore, in (10), the corresponding transmission costs are not included.

In the modeling of partial file storage, we consider a generic modeling that only considers file size, the most important parameter that does play a critical role in caching performance. In fact, Some other important information can also be introduced to refine the current modeling, such as considering the storage characteristics of different file types, i.e., the priority storage of thumbnails and other information for image files. The requested portion of the file content starts

from scratch by default. However, it is important to emphasize that the cached file contents are filtered based on the content popularity, not necessarily the head content of the large file, but most likely the key content in the middle.

D. Proposed DDQN-Based Distributed Edge Caching Algorithm with Content Recommendation

To speed up the convergence and reduce the impact of the curse of dimensionality in RL, we now propose a DRL-based distributed edge caching algorithm to find the optimal caching policy π^*.

DRL utilizes a deep neural network $Q(\boldsymbol{s}, \boldsymbol{a}; \theta)$ with the weight parameter θ to estimate $Q^*(\boldsymbol{s}, \boldsymbol{a})$ [32]. Due to the advantages of DDQN, a variant of DRL architecture [47], we propose a DDQN-based distributed edge caching algorithm with content recommendation. The details are presented in Algorithm 1. To optimize the caching policy π, the F-AP continuously interacts with the environment to learn. The environment is the user request dataset constructed by our proposed user request model in part B of section III. In addition, the F-AP recommends contents with high user preference to users within its coverage area, which can affect the user request model in (2).

(1) ***ϵ-greedy method:*** The objective of our optimization is the maximization of the reward in (13), where a trade-off between exploitation and exploration is needed. The ϵ-greedy method can achieve this goal [22]. As presented in line 17 of Algorithm 1, the ϵ-greedy method is employed in the training of action selection to avoid local optima in our proposed algorithm, where the greedy factor $\epsilon \in (0, 1)$. It is exploration if an action $\boldsymbol{a}_t$ is selected with probability ϵ. Otherwise, it exploitation if $\boldsymbol{a}_t = \arg\max_{\boldsymbol{a}_t} Q(\boldsymbol{s}_t, \boldsymbol{a}_t; \theta)$.

(2) ***Target Q-network:*** The DDQN architecture uses two neural networks for learning [47]: the current Q-network (the action-value function) Q and the target Q-network (the target action-value function) $\hat{Q}$. The current Q-network $Q(\boldsymbol{s}, \boldsymbol{a}; \theta)$ is exploited to select actions and update the network parameter θ. The target Q-network $\hat{Q}(\boldsymbol{s}, \boldsymbol{a}; \theta^-)$ is exploited to calculate the target Q-value. After every K steps of training of the current Q-network, its network parameters θ will be assigned to the target Q-network to update the parameters θ^- for the target Q-network, which can accelerate the convergence and reduce the correlation between the current Q-value and the target Q-value.

(3) ***Experience Replay:*** In the DDQN architecture, the experience vector $[\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}'_t]^T$ is required to update and train the current Q-network $Q(\boldsymbol{s}, \boldsymbol{a}; \theta)$. The experience vector $[\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}'_t]^T$ obtained by the agent interacting with the environment is stored in the replay memory D each time step. When the current Q-network needs to be trained, an experience vector $[\boldsymbol{s}_j, \boldsymbol{a}_j, r_j, \boldsymbol{s}'_j]^T$ will be selected randomly, and lines 21-28 of Algorithm 1 show the update process of two network models. Random selection of experience vectors for network training can not only cut off the correlation between experience vectors, but also prevent the problem from falling into a local optimum. Besides, in comparison with the DQN algorithm [32], the calculation of the target Q-value and the action selection are decoupled in the updating process to avoid any overestimate.

Algorithm 1 Proposed DDQN based Edge Caching Algorithm with Content Recommendation

1: Initialize replay memory D to capacity B;
2: Initialize target action-value function $\hat{Q}$ with weight $\theta^- = \theta$, action-value function Q with random weight θ;
3: Initialize $\mathcal{P}^0$, N_{max};
4: Initialize the number of cached files F_t in the storage space of the F-AP;
5: Fetch F_t files from the cloud server with the transfer learning method and record the file sizes;
6: Initialize $S_{\mathrm{av},t}$ based on (16) and initialize $S_{\max,t}$;
7: Split the F_t files with threshold $S_{\max,t}$, cache these files in the F-AP, and initialize initial state $\boldsymbol{s}_0$;
8: Set the cumulative number of requests for all cached files to 0, and set $U_0^{\mathrm{r}} = 0$;
9: **for** episode $= 0, 1, \ldots$ **do**
10: **for** time slot $t = 0, 1, \ldots, T$ **do**
11: U_t^{a} users newly arrive at the F-AP coverage, fetch each user preference vector $\boldsymbol{p}_{t,u}$ from $\mathcal{P}^0$ with some adjustments, and set the inactiveness parameters;
12: F-AP recommends contents, obtain θ_t and $\mathbf{rec}_t$;
13: $\mathbf{rec}_t$ affects $\mathcal{P}_t$ based on (2);
14: **for** user $u = 1, 2, \ldots, U_t$ **do**
15: Get $p^{\mathrm{l}}_{t,u}, p^{\mathrm{i}}_{t,u}, \mathrm{req}_{t,u}, N_{t,u}$;
16: the uth user requests files, $\mathrm{req}_{t,u}$ affects $\boldsymbol{p}_{t,u}$;
17: **end for**
18: Obtain U_{t+1}^{r} remaining users according to $\mathcal{P}_t^{\mathrm{l}}$, record their current user preferences;
19: Obtain $m(t)$ and $\mathcal{M}$ according to $\mathrm{req}_{t,u}$, θ_t;
20: Record request frequencies of the cached files in the storage space of the F-AP according to $\mathrm{req}_{t,u}$, θ_t, save the cached files in reverse order;
21: According to ϵ-greedy method, select a random action $\boldsymbol{a}_t$ with probability ϵ, otherwise select $\boldsymbol{a}_t = \arg\max_{\boldsymbol{a}_t} Q(\boldsymbol{s}_t, \boldsymbol{a}_t; \theta)$;
22: Execute action $\boldsymbol{a}_t$, update the cached files according to $a(t)$, $m(t)$, $\mathcal{M}$, get $\boldsymbol{s}'_t$;
23: Calculate reward r_t based on (13);
24: Save experience vector $[\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}'_t]^T$ in D;
25: Set $\boldsymbol{s}_t = \boldsymbol{s}'_t$;
26: Sample random experience vectors $\left[\boldsymbol{s}_j, \boldsymbol{a}_j, r_j, \boldsymbol{s}'_j\right]^T$ from D;

$$y_j = \begin{cases} r_j, \text{if time slot terminates at step } j+1; \\ r_j + \gamma \hat{Q}\big(\boldsymbol{s}'_j, \arg\max\limits_{\boldsymbol{a}'_j} Q(\boldsymbol{s}'_j, \boldsymbol{a}_j; \theta); \theta^-\big), \\ \qquad\qquad\qquad\qquad \text{otherwise}; \end{cases}$$

27: Execute a gradient descent step on $\big(y_j - Q(\boldsymbol{s}_j, \boldsymbol{a}_j; \theta)\big)^2$ with respect to the network parameter θ;
28: Reset $\hat{Q} = Q$ every K steps;
29: **end for**
30: Update $S_{\mathrm{av},t}$, $S_{\max,t}$ and F_t;
31: Update the current cached files of the F-AP;
32: Clear the state space $\mathcal{S}$ and initialize the initial state $\boldsymbol{s}_0$;
33: Clear all experience vectors in the replay memeory D.
34: **end for**

Computation Complexity: Regarding the complexity of the proposed DDQN-based edge caching algorithm, two aspects need to be examined, i.e., experience vectors and back propagation. Assuming that there exist K experience vectors in the experience pool, we have a complexity of $\mathcal{O}(K)$. Let x

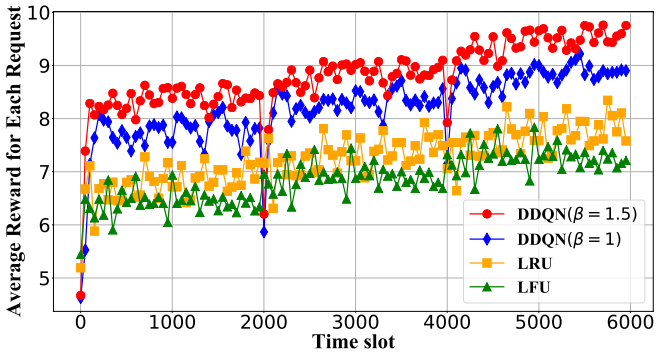


Fig. 4. Average reward for each request versus time slot for different caching algorithms.

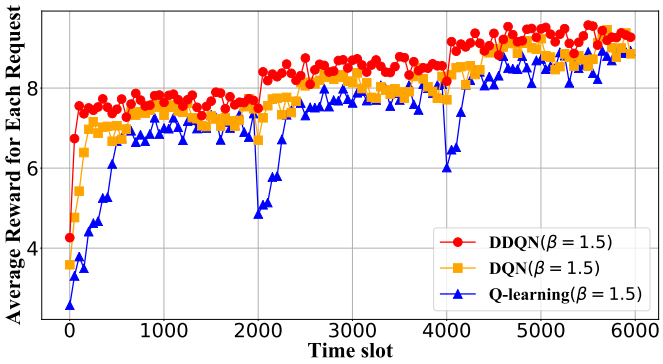


Fig. 5. Average reward for each request versus time slot for different RL caching algorithms.

represent the number of layers and y represent the number of experience vectors in a layer. It could take $\mathcal{O}(Lxyz)$ time to train the parameters by using gradient descent and back propagation. Here, L represents the number of randomly selected experience vectors, and z represents the number of iterations. Especially, K experience vectors are stored in the experience pool, and the space complexity of $\mathcal{O}(Kxy)$ is required to solve the parameter storage problem of the DDQN algorithm. Similar to [48], the analysis of computational complexity verifies that the proposed algorithm is lightweight and can be deployed to the edge device F-AP easily.

In general, the proposed algorithm can better adapt to dynamic user demands by learning from interactions with the request dataset. Content recommendation reduces the heterogeneity of user requests, and the caching efficiency of the original caching strategy has also been improved accordingly. It is also possible for the proposed algorithm to utilise the available prior knowledge to speed up training and reduce computational costs.

V. Simulation Results

The performance of our proposed edge caching algorithm is evaluated through simulations. The cloud server has $C = 50000$ files of different sizes, whose indexes form the file index library $\mathcal{C}$. The storage capacity C_{f} of each F-AP is set to 1000 unit size and the initial number of cached files in the F-AP is set to $F_t = 50$. A training period has 2000 time slots. The mean values of the elements in $\mathcal{P}_t^{\mathrm{l}}$ and $\mathcal{P}_t^{\mathrm{i}}$ are set to 0.5 and 0.1, respectively. In addition, we set: $G = 15$, $H = 5$, $\lambda = 30$, $N_{\max} = 3$, $\mu = 0.05$, $s = 0.1$, $\eta = 1.2$, $b = 1$, and $\beta = 1.5$. The proposed DDQN model adopts a three-layer fully connected network, in which the number of hidden layer neurons is 128 and the learning rate is 0.005. Some existing algorithms have been selected as benchmarks, namely the least recently used (LRU) method [49], the least frequently used(LFU) method, the Q-learning-based algorithm, the DQN-based algorithm, and our proposed algorithm without content recommendation ($\beta = 1$).

In Fig. 4, we plot the average rewards (net profits) over three training periods for each user request of the proposed algorithm (DDQN ($\beta = 1.5$)) and the three benchmark algorithms (LFU, LRU, the proposed algorithm with $\beta = 1$ (DDQN ($\beta = 1$)) , i.e., our proposed algorithm without considering the content recommendation). We can observe that, compared with LFU and LRU, our proposed algorithm ($\beta = 1.5$) has a clear advantage in improving average reward after three training cycles, and offers an increase of 29.7%. The reason is that the proposed algorithm can discriminate and reduce the 'redundant' updates of the F-AP triggered by infrequent user requests. Since a training period has 2000 time slots, it can be observed that the simulation curves related to deep reinforcement learning (DDQN ($\beta = 1$) and DDQN ($\beta = 1.5$)) all have a relatively obvious process of restarting training at every 2000 time slots. That is, the proposed algorithm can adapt to the situation of inconsistent file sizes in cloud servers, and the current cache capacity occupied by each cached file in the F-AP can be dynamically adjusted according to recent user requests. We can also observe that the proposed algorithm has a faster convergence speed and a better performance in comparison with DDQN ($\beta = 1$) which does not consider content recommendation. Besides, our proposed algorithm shows small fluctuations after convergence because the F-AP constantly interacts with the environment to learn and optimize the caching strategy to satisfy the fluctuant demands of users.

In Fig. 5, we compare the average reward of each user request between our proposed algorithm and two classic reinforcement learning algorithms, i.e., DQN and Q-learning. We can observe that all the simulation curves follow an identical pattern in restarting training for every 2000 time slots. We can also observe that in terms of convergence speed and average reward, the two benchmark algorithms are significantly worse than our proposed algorithm, which means that the probability of local optima can be greatly reduced. Moreover, the Q-learning-based algorithm is worse than the other two algorithms based on deep neural networks (DQN and DDQN) in terms of convergence speed. The reason is that DRL employs the neural network $Q(\boldsymbol{s}, \boldsymbol{a}; \theta)$ to simulate $Q^*(\boldsymbol{s}, \boldsymbol{a})$, which shortens the time for the algorithm to find the optimal value.

In Fig. 6, we plot the average rewards of each user request for the proposed algorithm and the three benchmark algorithms (LFU, LRU, and DDQN ($\beta = 1$)) with different storage

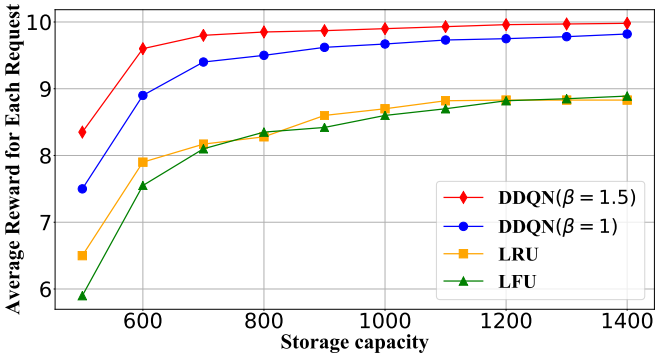


Fig. 6. Average reward for each request versus the storage capacity for different caching algorithms.

capacity C_{f}. In order to improve the reliability and fairness of the experimental results, when each algorithm converges after three training periods, we consider the mean of the average reward for each request. The total cache size increases from $C_{\mathrm{f}} = 600$ unit size to $C_{\mathrm{f}} = 1400$ unit size. We can see that the average rewards increase as storage capacity increases for four algorithms. We can also observe that our proposed algorithm has a larger average reward compared with two traditional algorithms, i.e., LRU and LFU, and in the case of smaller storage capacity, our proposed algorithm also has better performance.

VI. Conclusion

In this paper, we have studied the distributed edge caching problem in F-RANs with dynamic content recommendation. In particular, we have proposed a more realistic personalized time-varying user request model to describe the fluctuant user demands. To maximize the net profit of each F-AP, we have proposed a DDQN-based distributed edge caching algorithm with content recommendation to find the optimal caching policy. Simulation results have shown that the proposed algorithm offers a better caching performance in comparison with the traditional methods. The proposed algorithm can adapt to the situation of inconsistent file sizes in the cloud server, and the storage space occupied by each cached file in the F-AP can be dynamically adjusted according to user requests. Finally, content recommendation can also enhance the performance of the original caching policy to a certain degree. In subsequent studies, we plan to consider introducing more parameters to improve the current modeling, such as storage characteristics of different file types, the wireless channel and its impediments, and compare with more machine learning based caching algorithms.

References

[1] X. You, C. Wang, J. Huang, and et al., "Towards 6G wireless communication networks: Vision, enabling technologies, and new paradigm shifts," *Sci. China-Inf. Sci.*, vol. 64, no. 1, pp. 1–74, Jan. 2021.

[2] E. Zeydan, E. Bastug, M. Bennis, and et al., "Big data caching for networking: Moving from cloud to edge," *IEEE Commun. Mag.*, vol. 54, no. 9, pp. 36–42, Sept. 2016.

[3] Z. Chang, L. Lei, Z. Zhou, and et al., "Learn to cache: Machine learning for network edge caching in the big data era," *IEEE Wirel. Commun.*, vol. 25, no. 3, pp. 28–35, Jun. 2018.

[4] Y. Huang, X. Song, F. Ye, Y. Yang, and X. Li, "Fair and efficient caching algorithms and strategies for peer data sharing in pervasive edge computing environments," *IEEE Transactions on Mobile Computing*, vol. 19, no. 4, pp. 852–864, Apr. 2020.

[5] J. Ni, X. Lin, and X. S. Shen, "Efficient and secure service-oriented authentication supporting network slicing for 5G-enabled IoT," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 644–657, Mar. 2018.

[6] Y. Jiang, M. Ma, M. Bennis, and et al., "User preference learning-based edge caching for fog radio access network," *IEEE Trans. Commun.*, vol. 67, no. 2, pp. 1268–1283, Feb. 2019.

[7] Y. Jiang, W. Huang, M. Bennis, and et al., "Decentralized asynchronous coded caching design and performance analysis in fog radio access networks," *IEEE Trans. Mob. Comput.*, pp. 1–12, Jan. 2019.

[8] Y. Jiang, H. Feng, F.-C. Zheng, D. Niyato, and X. You, "Deep learning-based edge caching in fog radio access networks," *IEEE Transactions on Wireless Communications*, vol. 19, no. 12, pp. 8442–8454, Sep. 2020.

[9] Y. Jiang, A. Peng, C. Wan, and et al., "Analysis and optimization of cache-enabled fog radio access networks: Successful transmission probability, fractional offloaded traffic and delay," *IEEE Trans. Veh. Technol.*, vol. 69, no. 5, pp. 5219–5231, May. 2020.

[10] A. Peng, Y. Jiang, M. Bennis, and et al., "Performance analysis and caching design in fog radio access networks," in *2018 IEEE Globecom Workshops*, Dec. 2018, pp. 1–6.

[11] Y. Jiang, C. Wan, M. Tao, and et al., "Analysis and optimization of fog radio access networks with hybrid caching: Delay and energy efficiency," *IEEE Trans. Wireless Commun.*, vol. 20, no. 1, pp. 69–82, Jan. 2021.

[12] J. Li, Y. Chen, Z. Lin, and et al., "Distributed caching for data dissemination in the downlink of heterogeneous networks," *IEEE Trans. Commun.*, vol. 63, no. 10, pp. 3553–3568, Oct. 2015.

[13] Y. Jiang, X. Cui, M. Bennis, and F. C. Zheng, "Cooperative caching in fog radio access networks: A graph-based approach," *IET Commun.*, vol. 13, no. 20, pp. 3519–3528, Nov. 2019.

[14] X. Cui, Y. Jiang, X. Chen, F. Zheng, and X. You, "Graph-based cooperative caching in fog-ran," in *2018 Int. Conf. Comput. Netw. Commun.*, Mar. 2018, pp. 166–171.

[15] Q. Chen, F. R. Yu, T. Huang, R. Xie, J. Liu, and Y. Liu, "Joint resource allocation for software-defined networking, caching, and computing," *IEEE/ACM Transactions on Networking*, vol. 26, no. 1, pp. 274–287, Feb. 2018.

[16] Y. Jiang, Y. Hu, M. Bennis, and et al., "A mean field game-based distributed edge caching in fog radio access networks," *IEEE Trans. Commun.*, vol. 68, no. 3, pp. 1567–1580, Mar. 2020.

[17] Y. Jiang, M. Ma, M. Bennis, and et al., "A novel caching policy with content popularity prediction and user preference learning in fog-ran," in *2017 IEEE Globecom Workshops*, Dec. 2017, pp. 1–6.

[18] H. Feng, Y. Jiang, D. Niyato, and et al., "Content popularity prediction via deep learning in cache-enabled fog radio access networks," in *2019 IEEE Glob. Commun. Conf.*, Dec. 2019, pp. 1–6.

[19] Q. F. P. F. J. W. Q. Wu, Y. Zhao and C. Zhang, "Mobility-aware cooperative caching in vehicular edge computing based on asynchronous federated and deep reinforcement learning," *IEEE Journal of Selected Topics in Signal Processing*, vol. 17, no. 1, pp. 66–81, Jan. 2023.

[20] L. Lu, Y. Jiang, M. Bennis, and et al., "Distributed edge caching via reinforcement learning in fog radio access networks," in *2019 IEEE 89th Veh. Technol. Conf.*, Apr. 2019, pp. 1–6.

[21] J. Yan, Y. Jiang, F. Zheng, and et al., "Distributed edge caching with content recommendation in Fog-RANs via deep reinforcement learning," in *2020 IEEE Int. Conf. Commun. Workshops*, Jun. 2020, pp. 1–6.

[22] D. Liu and C. Yang, "A learning-based approach to joint content caching and recommendation at base stations," in *2018 IEEE Glob. Commun. Conf.*, Dec. 2018, pp. 1–7.

[23] A. Sadeghi, F. Sheikholeslami, and G. B. Giannakis, "Optimal and scalable caching for 5G using reinforcement learning of space-time popularities," *IEEE J. Sel. Top. Signal Process.*, vol. 12, no. 1, pp. 180–190, Feb. 2018.

[24] X. K. et al., "Deep reinforcement learning-based energy-efficient edge computing for internet of vehicles," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 9, pp. 6308–6316, Sept. 2022.

[25] C. F. et al., "Deep-reinforcement-learning-based resource allocation for content distribution in fog radio access networks," *IEEE Internet of Things Journal*, vol. 9, no. 18, pp. 16 874–16 883, Sept. 2022.

[26] H. P. Linder and S. D. Johnson, "The netflix recommender system: Algorithms, business value, and innovation," *ACM Trans. Manag. Inf. Syst.*, vol. 6, no. 4, pp. 1–19, Dec. 2015.

[27] Z. Lin and W. Chen, "Content pushing over multiuser miso downlinks with multicast beamforming and recommendation: A cross-layer approach," *IEEE Trans. Commun.*, vol. 67, no. 10, pp. 7263–7276, Jul. 2019.

[28] K. Guo, C. Yang, and T. Liu, "Caching in base station with recommendation via Q-Learning," in *2017 IEEE Wirel. Commun. Netw. Conf.*, Mar. 2017, pp. 1–6.

[29] L. E. Chatzieleftheriou, M. Karaliopoulos, and I. Koutsopoulos, "Jointly optimizing content caching and recommendations in small cell networks," *IEEE Trans. Mob. Comput.*, vol. 18, no. 1, pp. 125–138, Apr. 2019.

[30] D. Liu and C. Yang, "A deep reinforcement learning approach to proactive content pushing and recommendation for mobile users," *IEEE Access*, vol. 7, pp. 83 120–83 136, Jul. 2019.

[31] K. Guo and C. Yang, "Temporal-spatial recommendation for caching at base stations via deep reinforcement learning," *IEEE Access*, vol. 7, pp. 58 519–58 532, May. 2019.

[32] V. Mnih, K. Kavukcuoglu, D. Silver, and et al., "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[33] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *2010 Proceedings IEEE Conf. Comput. Commun.*, Mar. 2010, pp. 1–9.

[34] B. Blaszczyszyn and A. Giovanidis, "Optimal geographic caching in cellular networks," in *2015 IEEE Int. Conf. Commun.*, Jun. 2015, pp. 3358–3363.

[35] E. Zeydan, E. Bastug, M. Bennis, and et al., "Big data caching for networking: Moving from cloud to edge," *IEEE Commun. Mag.*, vol. 54, no. 9, pp. 36–42, Sept. 2016.

[36] S. Zhang, P. He, K. Suto, and et al., "Cooperative edge caching in user-centric clustered mobile networks," *IEEE Trans. Mob. Comput.*, vol. 17, no. 8, pp. 1791–1805, Aug. 2018.

[37] A. Sengupta, R. Tandon, and O. Simeone, "Fog-aided wireless networks for content delivery: Fundamental latency tradeoffs," *IEEE Trans. Inf. Theory*, vol. 63, no. 10, pp. 6650–6678, Oct. 2017.

[38] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," *IEEE Trans. Inf. Theory*, vol. 60, no. 5, pp. 2856–2867, May. 2014.

[39] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *2018 - IEEE Conf. Comput. Commun.*, Apr. 2018, pp. 207–215.

[40] A. Sadeghi, F. Sheikholeslami, and G. B. Giannakis, "Optimal and scalable caching for 5G using reinforcement learning of space-time popularities," *IEEE J. Sel. Top. Signal Process.*, vol. 12, no. 1, pp. 180–190, Feb. 2018.

[41] M. Hefeeda and O. Saleh, "Traffic modeling and proportional partial caching for peer-to-peer systems," *IEEE/ACM Trans. Netw.*, vol. 16, no. 6, pp. 1447–1460, Dec. 2008.

[42] J. Yang, Y. Qiao, X. Zhang, and et al., "Characterizing user behavior in mobile internet," *IEEE Trans. Emerg. Top. Comput.*, vol. 3, no. 1, pp. 95–106, Mar. 2015.

[43] L. P. Kaelbling, M. L. Littman, and et al., "An introduction to reinforcement learning," *IEEE Trans. Neural Netw.*, vol. 16, no. 1, pp. 285–286, Jan. 2005.

[44] X. Wang, M. Chen, T. Taleb, and et al., "Cache in the air: Exploiting content caching and delivery techniques for 5g systems," *IEEE Commun. Mag.*, vol. 52, no. 2, pp. 131–139, Feb. 2014.

[45] B. N. Bharath, K. G. Nagananda, and H. V. Poor, "A learning-based approach to caching in heterogenous small cell networks," *IEEE Trans. Commun.*, vol. 64, no. 4, pp. 1674–1686, Apr. 2016.

[46] M. Leconte, G. Paschos, L. Gkatzikis, and et al., "Placing dynamic content in caches with small population," in *2016 - The 35th Annual IEEE Int. Conf. Comput. Commun.*, Apr. 2016, pp. 1–9.

[47] H. V. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," *Comput. Sci.*, Sept. 2015.

[48] B. Chen, L. Liu, M. Sun, and et al., "Iotcache: Toward data-driven network caching for internet of things," *IEEE Internet Things J.*, vol. 6, no. 6, pp. 10 064–10 076, Aug. 2019.

[49] H. Ahlehagh and S. Dey, "Video caching in radio access network: Impact on delay and capacity," in *2012 IEEE Wirel. Commun. Netw. Conf.*, Apr. 2012, pp. 2276–2281.

Jie Yan received the M.S. degree in communications and information systems from Southeast University, Nanjing, China.

Her research interests include radio resource management and edge caching.

Min Zhang received the M.S. degree in communications and information systems from Southeast University, Nanjing, China.

His research interests include radio resource management and edge caching.

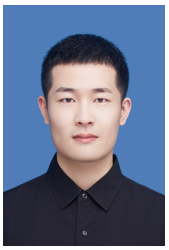
Yanxiang Jiang (S'03-M'07-SM'18) received the B.S. degree in electrical engineering from Nanjing University, Nanjing, China, in 1999 and the M.S. and Ph.D. degrees in communications and information systems from Southeast University, Nanjing, China, in 2003 and 2007, respectively.

Dr. Jiang was a Visiting Scholar with the Signals and Information Group, Department of Electrical and Computer Engineering, University of Maryland at College Park, College Park, MD, USA, in 2014. He is currently an Associate Professor with the National Mobile Communications Research Laboratory, Southeast University, Nanjing, China. His research interests are in the area of broadband wireless mobile communications, covering topics such as machine learning for wireless communications, edge caching, radio resource allocation and management, fog radio access networks, small cells and heterogeneous networks, cooperative communications, green communications, device to device communications, and massive MIMO.

Fu-Chun Zheng obtained the BEng (1985) and MEng (1988) degrees in radio engineering from Harbin Institute of Technology, China, and the PhD degree in Electrical Engineering from the University of Edinburgh, UK, in 1992.

From 1992 to 1995, he was a post-doctoral research associate with the University of Bradford, UK, Between May 1995 and August 2007, he was with Victoria University, Melbourne, Australia, first as a lecturer and then as an associate professor in mobile communications. He was with the University of Reading, UK, from September 2007 to July 2016 as a Professor (Chair) of Signal Processing. He has also been a distinguished adjunct professor with Southeast University, China, since 2010. Since August 2016, he has been with Harbin Institute of Technology (Shenzhen), China, as a distinguished professor, and the University of York, UK. He has been awarded two UK EPSRC Visiting Fellowships - both hosted by the University of York (UK): first in August 2002 and then again in August 2006. Over the past two decades, Dr Zheng has also carried out many government and industry sponsored research projects - in Australia, the UK, and China. He has been both a short term visiting fellow and a long term visiting research fellow with British Telecom, UK. Dr Zheng's current research interests include multiple antenna systems, green communications, ultra-dense networks, and ultra-reliable low latency communications (URLLC).

He has been an active IEEE member since 1995. He was an editor (2001-2004) of IEEE Transactions on Wireless Communications. In 2006, Dr Zheng served as the general chair of IEEE VTC 2006-S, Melbourne, Australia (www.ieeevtc.org/vtc2006spring) - the first ever VTC held in the southern hemisphere in VTCs history of six decades. More recently he was the executive TPC Chair for VTC 2016-S, Nanjing, China (the first ever VTC held in mainland China: www.ieeevtc.org/vtc2016spring).



Qi Chang received the M.S. degree in communications and information systems from Southeast University, Nanjing, China.

His research interests include radio resource management and edge caching.

Xiaohu You (SM'11-F'12) received the B.S., M.S., and Ph.D. degrees in electrical engineering from Southeast University, Nanjing, China, in 1982, 1985, and 1988, respectively. Since 1990, he has been working with National Mobile Communications Research Laboratory at Southeast University, where he holds the ranks of professor and director. He is the Chief of the Technical Group of China 3G/B3G Mobile Communication R&D Project. His research interests include mobile communications, adaptive signal processing, and artificial neural networks, with applications to communications and biomedical engineering.

Dr. You was a recipient of the Excellent Paper Prize from the China Institute of Communications in 1987; the Elite Outstanding Young Teacher award from Southeast University in 1990, 1991, and 1993; and the National Technological Invention Award of China in 2011. He was also a recipient of the 1989 Young Teacher Award of Fok Ying Tung Education Foundation, State Education Commission of China.