DOI:
10.1109/TMC.2024.3382824

Document Version
Peer reviewed version

A Unified Federated Deep Q Learning Caching Scheme for Scalable Collaborative Edge Networks

Ming Zhao, *Student Member, IEEE,* and Mohammad Reza Nakhai, *Senior Member, IEEE*

***Abstract*—Edge caching-enabled networks can efficiently alleviate data traffic and improve quality of service. However, effectively adapting to users' heterogeneous requests and coordinating among multiple edge servers remains a challenge. In this paper, we address the collaborative cache update and request delivery problem in an edge caching system, aiming to minimize the long-term average system cost under uncertainties of users' heterogeneous demands and dynamic content popularity. To overcome the curse of dimensionality, we decompose the formulated problem into two subproblems: the coordinated proactive cache updating and local request processing. Next, we propose a unified federated deep Q learning (DQL) caching scheme to tackle and coordinate these two subproblems. Particularly, our scheme features a scalable DQL approach with a two-phase action selection procedure to learn the heterogeneous user requests across distributed servers in an online manner. Furthermore, we develop a federated learning (FL)-empowered training process to improve coordination among multiple servers, in which a Thompson sampling (TS)-based algorithm is introduced for smart server selection. We evaluate the performance of our proposed caching scheme in both small-scale and large-scale scenarios through comprehensive experiments, which highlights the advantages of the proposed scheme in terms of caching performance, scalability and robustness.**

***Index Terms*—Collaborative edge caching, deep reinforcement learning, federated learning, smart server selection.**

I. Introduction

With the development of intelligent mobile devices, the demand of multimedia services and latency-critical services has been growing dramatically, which inevitably leads to serious mobile traffic [1], [2]. To cope with this challenge, a new trend is happening with the cloud functionalities increasingly shifting towards the network edges. Mobile edge computing (MEC) is one of the emerging techniques, which enables the capabilities of computing, caching and communication at the network edges that are in close proximity to the end users [3]–[5]. Consequently, MEC is capable of supporting computation-intensive and latency-sensitive mobile applications or multimedia services for the resource-constrained mobile devices, so as to reduce the communication delay and improve the quality of service/experience (QoS/QoE).

As one significant feature of MEC, edge caching provides the ability to lower the retrieving latency by caching popular files at edge servers. Particularly, it is shown that a plethora of mobile data traffic is caused by duplicate downloads of popular video files within a local area and a certain period of time [**?**]. From the perspective of the overall caching system, caching repeatedly requested files in a targeted area will greatly reduce the transmission cost and alleviate the mobile traffic. Nonetheless, current edge caching in mobile edge networks still faces two challenges: *heterogeneous user requests and coordination among multiple edge servers*.

On the one hand, each edge server can only cache a small fraction of files due to its limited storage capacity. The massive users' requests may not be satisfied directly by the associated servers, which decreases the cache hit rate and affects the QoE of users. On the other hand, different users' requests are generally highly heterogeneous. Even the same user may follow different content request patterns at different time periods, which further aggravates the request heterogeneity. As a result, the caching policy should be specialized for each edge server to cope with the time-varying content requests under the storage constraint. Meanwhile, edge servers need to develop the caching policies in a collaborative way to improve the overall caching performance, e.g., reduce overall transmission cost and improve cache hit rate.

To tackle the above challenges, advanced approaches based on machine learning techniques have been flourishing. Especially, reinforcement learning (RL) provides an efficient and effective solution for handling edge caching problems. It can learn to capture "hidden insights" of massive user requests in dynamic and complex environment, consequently optimizing the caching policies [6]. Moreover, deep reinforcement learning (DRL) [7], [8], which integrates deep neural networks into RL, has been widely applied in edge caching. Nevertheless, DRL suffers performance degradation due to the curse of dimensionality in large-scale edge caching networks. This motivates our exploration of a novel approach to address the high-dimensional challenges of DRL in the context of large-scale edge caching networks.

In this paper, we investigate the collaborative cache update and request delivery problem in an edge caching system, which jointly considers the challenges of heterogeneous user requests, coordination among multiple edge servers, as well as network scalability. We propose a unified federated deep Q learning (DQL) caching scheme, where each edge server optimizes its own caching policy while coordinating with its proximal servers to minimize overall system cost and improve cache hit rate, under the assumption that the prior knowledge of the dynamic content popularity and the heterogeneous user preferences are unknown in advance. In particular, the proposed scheme integrates the actions of DQL [9] and federated learning (FL) [10] to jointly enable coordinated intelligent caching at distributed servers.

Our distinct contributions are outlined as follows:

- **Integration of FL and DQL:** We develop a DQL approach to learn the heterogeneous users' requests locally, in order to adaptively update the caching decisions at

distributed servers. To coordinate training among DQL models at a global server, we adopt FL concept and integrate it into local DQL training processes to further improve the proposed distributed caching scheme.

- **Scalability of the edge caching network:** To deal with the large dimension in action space and to prepare for the scalability of the caching network, we propose a scalable DQL approach by re-designing the output layer of the DQL model. A two-phase action selection procedure is further developed to better balance exploitation and exploration.
- **Signaling overhead reduction:** To reduce the signaling overhead among edge servers and the global server, a Thompson sampling (TS)-based agent selection method is introduced to predictively search for the optimal server set in the global model training process. Moreover, the proposed DQL model is partitioned into a common-network part and the output layer part, and only the common-network parameters are sent to the global server for the global model training, while the output layer parameters are trained locally to preserve the user preferences at individual servers.

The rest of this paper is organized as follows. Section II summarizes the related work. Section III presents the system architecture and problem formulation. In Section IV, we propose a unified federated DQL edge caching scheme to address the formulated problem. Next, we evaluate the performance of the proposed scheme in Section V, and conclude the paper in Section VI.

II. Related Work

The design of caching policies is primarily influenced by content popularity. Two commonly used cache replacement policies, the least recently used (LRU) [11] and the least frequently used (LFU) [12], are developed to update the cached files from time to time according to the instantaneous user requests. Although these two conventional policies are of low computational complexity and easy to implement, they cannot deal with large-scale complex situations, where the network conditions are changing dynamically.

At this point, many caching policies based on machine learning have been developed recently for improving the QoS/QoE [1], [13]. To maximize the cache hit rate, authors in [14] proposed an online distributed cache replacement algorithm based on TS. In [15], authors considered a two-level network caching scenario, where a parent node is connected to multiple distributed leaf nodes to serve users. Then a hyper DQL framework was proposed to deal with the dynamic evolution of file requests. In [16], a multi-agent actor-critic RL framework was introduced. Each base station (BS), as one actor, makes its own caching decision based on a partially observable Markov Decision Process while competing for chance to serve the covered users. Meanwhile, each BS cooperates with others to minimize the overall transmission delay. However, these studies were built on the assumption of a homogeneous distribution of users' requests, which is unrealistic in real-world scenarios, limiting their applicability.

In [17], authors introduced a cooperative edge caching framework based on double DQL to minimize the long-term average content fetching delay, under the assumption that content popularity distribution is unknown a priori. Authors in [18] decomposed the caching problem in MEC system into user-side cache optimization problem and BS-side cache optimization problem, and then developed a joint cache and delivery policy under the assumption that the policy on one side (either user or BS) is fixed while optimizing the policy on the other side. DQL was hereafter applied on BS to optimize the BS-side caching policy. Nonetheless, the caching policy in this study was designed for each BS independently, while the cooperation among BSs was not investigated. With the aim of maximizing cache hit rate and minimizing the transmission delay, authors in [19] proposed two deep actor-critic RL based policies for centralized and decentralized content caching, respectively. However, the DRL based algorithms (e.g., DQL and actor-critic) in these existing studies [16]–[19] suffer performance degradation due to the curse of dimensionality, and hence lack scalability.

To deal with the issue of dimensionality, authors in [20] proposed a soft actor-critic based algorithm for discrete space. But higher transmission cost will be required for this algorithm to achieve frequent model parameter exchange between the cloud server and edge servers while training the learning models. By leveraging soft actor-critic and FL, a decentralized recommendation-enabled edge caching framework was proposed in [21]. But the proposed caching policy for each server can only replace at most one content in each time slot, which significantly limits the caching efficiency. Moreover, the cooperation among edge servers, e.g., content sharing with other proximal servers, was not considered in this study.

III. Problem Formulation

In this section, we first introduce the system architecture in subsection III-A. Next, we define the variables and cost functions to measure the QoS in subsection III-B. Finally, we formulate the collaborative cache update and request delivery problem in subsection III-C.

A. System architecture

As illustrated in Fig. 1, we consider a collaborative edge caching system consisting of one cloud center that stores a total number of F files, and M edge servers. Let $\mathcal{F} = \{1, 2, ..., f, ..., F\}$ and $\mathcal{M} = \{1, 2, ..., M\}$, respectively, denote the set of files and the set of edge servers. Each file $f \in \mathcal{F}$ is of size n_f. Each edge server is co-located with one BS and equipped with a limited cache storage of size $N_m, m \in \mathcal{M}$, and each edge server is capable of migrating cached files to its proximal edge servers. Specifically, the edge server m covers U_m users, where $U_m \in \mathcal{U}_m = \{1, 2, ..., U_m\}, m \in \mathcal{M}$, and each user is served by its associated edge server only.

Let $\mathcal{T} = \{1, 2, ..., T\}$ denote a set of discrete time slots, where T is the finite time horizon. Within a time slot $t \in \mathcal{T}$, each user requests for at most one file, which must be served before the end of the current time slot. Let $D^t_{m,u} = f \in \bar{\mathcal{F}}, m \in \mathcal{M}, u \in \mathcal{U}_m$ denote the request of user u associated

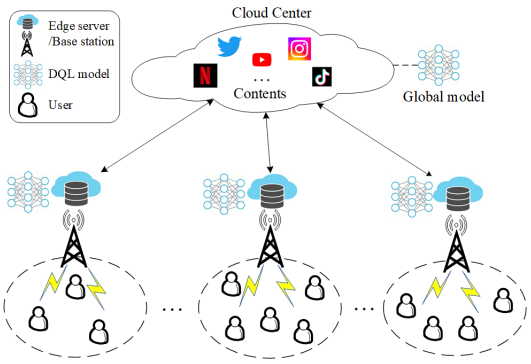

Fig. 1. System model.

with edge server m at time slot t, where $\bar{\mathcal{F}} = \mathcal{F} \bigcup \{0\}$, and $D^t_{m,u} = 0$ indicates that there is no request by user u. We assume that the demand of each user follows a Markov chain, which captures both the file popularity and temporal file correlations [22]. Let the evolution of user demand be modelled as $D^{t+1}_{m,u} = p^{m,u}_{i,j} \times D^t_{m,u}$, where $p^{m,u}_{i,j}$ is the transition probability of requesting file $j \in \bar{\mathcal{F}}$ at time slot $t+1$ given that user u has requested file $i \in \bar{\mathcal{F}}$ at time slot t. Details of user request model are introduced in Sec.V. In this paper, we consider a realistic scenario with heterogeneous content requests, which means $p^{m,u}_{i,j}$ for different users are diverse and dynamic. Besides, both of the file popularity and temporal correlations are unknown in advance.

The content demands of users $D^t_{m,u}, \forall u \in \mathcal{U}_m, \forall m \in \mathcal{M}$ can be satisfied by the associated edge server directly, or by retrieving from one of the caches of other edge servers or from the cloud center. In the proposed model, each edge server can only exchange information with its proximal edge servers via communications between the associated BSs, which could be either an exchange of the respective caching status or sharing files with one another. We denote the set of the proximal edge servers with respect to the local edge server m by $\mathcal{M}_m$, where $|\mathcal{M}_m| < |\mathcal{M}|$, and $|\mathcal{M}_m|, |\mathcal{M}|$ denote the number of edge servers in the sets $\mathcal{M}_m$ and $\mathcal{M}$, respectively.

Let the binary variable $a^t_{m,f} \in \{0,1\}$ indicate the caching state of file f, where $a^t_{m,f} = 1$ and $a^t_{m,f} = 0$, respectively, stand for the cached and otherwise states of the file f in server m at time slot t. Similarly, we let the binary variable $h^t_{m,m',f} \in \{0,1\}, \forall m' \neq m, m' \in \mathcal{M}_m$, indicate the fetching state of file f, where $h^t_{m,m',f} = 1$ and $h^t_{m,m',f} = 0$, respectively, stand for the fetched and otherwise states of the file f by server m from server m' at time slot t. Furthermore, the binary variable $h^t_{m,c,f} \in \{0,1\}$ indicates that the file f is retrieved by server m from the cloud, if $h^t_{m,c,f} = 1$, and otherwise, if $h^t_{m,c,f} = 0$.

B. System cost model

As introduced above, edge servers within close proximity can perform content caching and sharing in a collaborative way. Although collaborative caching may be resource-demanding, it could potentially improve cache hit rate and reduce transmission cost, compared with that of retrieving files from the remote cloud center. From the perspective of the mobile network operator, the system serving cost is a key indicator to measure the QoS, which includes proactive caching cost, reactive fetching cost and remote retrieving cost.

1) Proactive caching cost: The proactive content caching will incur a corresponding cost, which is calculated based on the caching space occupied by the files. So the cost of server m storing file f at time slot t is given by

$$P^t_{m,f} = \frac{\tau^{\text{cache}} n_f}{\tilde{D}^t_{m,f}}, \forall m \in \mathcal{M}, f \in \mathcal{F}, t \in \mathcal{T}, \tag{1}$$

where $\tau^{\text{cache}} > 0$ is the caching cost per unit file size, $\tilde{D}^t_{m,f} = \sum_{u \in \mathcal{U}_m} \mathbb{1}(D^t_{m,u} = f)$ represents the number of requested times for file f, received by server m at time slot t, and $\mathbb{1}(\cdot)$ denotes the indicator function. The factor $1/\tilde{D}^t_{m,f}$ in (1) indicates that the caching cost of a specific file will be lower if the file is repeatedly requested.

2) Reactive fetching cost: The cost of server m fetching file f from a neighbouring server m' at time slot t is given by

$$P^t_{m,m',f} = \frac{\tau^{\text{fet}}_{mm'} n_f}{\tilde{D}^t_{m,f}}, \forall m' \neq m, m \in \mathcal{M}, m' \in \mathcal{M}_m, f \in \mathcal{F}, t \in \mathcal{T}, \tag{2}$$

where $\tau^{\text{fet}}_{mm'} > 0$ is the unit fetching cost of file migration from a neighbouring server m' to the local server m. We further assume that $\tau^{\text{fet}}_{mm'_1} \neq \tau^{\text{fet}}_{mm'_2}, \forall m'_1 \neq m'_2, m'_1, m'_2 \in \mathcal{M}_m$. Similar to the proactive caching cost, the factor $1/\tilde{D}^t_{m,f}$ in (2) is applied to reduce the fetching cost if file f is requested by multiple users associated to server m.

3) Remote retrieving cost: Obviously, the QoS will be significantly affected by users who cannot obtain the requested file from any available edge servers. Let $P^t_{m,c,f}$ denote the cost of server m retrieving file f from remote cloud center at time slot t, expressed as

$$P^t_{m,c,f} = \tau^{\text{ret}} n_f \tilde{D}^t_{m,f}, \forall m \in \mathcal{M}, f \in \mathcal{F}, t \in \mathcal{T}, \tag{3}$$

where $\tau^{\text{ret}} > 0$ is the retrieving cost per unit file size. The factor $\tilde{D}^t_{m,f}$ in (3) is applied as a penalty, which means the retrieving cost will be higher in case multiple users request the same file f which is not cached locally. Given the fact that the corresponding cost is the largest for retrieving files from remote cloud center, we have $\tau^{\text{ret}} \gg \tau^{\text{fet}}_{mm'} > \tau^{\text{cache}}$.

C. Collaborative cache update and request delivery problem

Using the definitions and the cost functions developed above, we formulate the problem of the optimal cache update and request delivery as a long-term foresighted minimization of the overall system cost over a time horizon $t \in \mathcal{T}$ by jointly finding the binary decision vectors $\mathbf{a}^t_m$ and $\mathbf{h}^t_m$ for each and every server m as

$$\min_{\mathbf{a}^t_m, \mathbf{h}^t_m} \sum_{t \in \mathcal{T}} \sum_{f \in \mathcal{F}} P^t_{m,f} a^t_{m,f} + P^t_{m,m',f} h^t_{m,m',f} + P^t_{m,c,f} h^t_{m,c,f} \tag{4}$$

$$\text{s. t.} \sum_{f \in \mathcal{F}} n_f a^t_{m,f} \leq N_m, \quad \forall m \in \mathcal{M}, t \in \mathcal{T}, \tag{5}$$

$$a^t_{m,f} + h^t_{m,m',f} + h^t_{m,c,f} = 1, \forall m \in \mathcal{M}, t \in \mathcal{T}, f \in \mathcal{F}, \tag{6}$$

where $\mathbf{a}_m^t = \{a_{m,f}^t\}_{f\in\mathcal{F}}$ denotes the proactive cache updating decision vector, $\mathbf{h}_m^t = \{h_{m,m',f}^t, h_{m,c,f}^t\}_{f\in\mathcal{F}}$ denotes the content migration decision vector, (5) is the cache size constraint, and constraint (6) indicates that a request can be satisfied by one place only, either an edge server or the remote cloud.

However the problem in (4) heavily involves binary variables, and hence falls within the class of binary optimization problems, which is combinatorial and hard to solve by conventional means. It also involves evolution over time and aims at minimizing a long-term cost function without any prior knowledge of the statistical dynamics of the network, such as the unknown user request transition probabilities. In the sequel, we introduce reinforcement learning (RL) to capture the features of these environmental dynamics. We formulate the RL model for each server $m \in \mathcal{M}$ (i.e., an agent) in terms of state, action and reward.

State: An optimal caching policy for a server is expected to cache those files with the largest number of requesting times. Hence, the order of content requests does not affect caching optimization. Thus, let the state vector m at time slot t, $\mathbf{s}_m^t \in \mathcal{S}_m$, be defined as $\mathbf{s}_m^t = \left[\tilde{\mathbf{D}}_m^t, \mathbf{a}_m^{t-1}\right]^\top$, where $\tilde{\mathbf{D}}_m^t = \{\tilde{D}_{m,f}^t\}_{f\in\mathcal{F}}$ denotes the number of requests for each file in edge server m at the current time slot t, and $\mathbf{a}_m^{t-1} = \{a_{m,f}^{t-1}\}_{f\in\mathcal{F}}$ indicates the corresponding caching states of the files at the previous time slot $t-1$.

Action: The action vector $\mathbf{A}_m^t = \left[\mathbf{a}_m^t, \mathbf{h}_m^t\right]^\top$ for each edge server contains two component vectors, namely, the proactive cache updating action $\mathbf{a}_m^t$, and the content migration action $\mathbf{h}_m^t$.

Reward: According to problem (4), our optimization objective is to minimize the overall system serving cost in a long run. Thus, we define the instantaneous reward for the agent m at time slot t as the negative value of a sum of serving cost, i.e.,

$$r_m^t = -\sum_{f\in\mathcal{F}} P_{m,f}^t a_{m,f}^t + P_{m,m',f}^t h_{m,m',f}^t + P_{m,c,f}^t h_{m,c,f}^t. \quad (7)$$

IV. A Unified Federated Deep Q Learning Scheme

According to the RL model formulated in subsection III-C, the dimension of the action space is notably large, which leads to extensive exploration of the action-state space in RL. In this section, we propose a unified federated DQL caching scheme to overcome the curse of dimensionality and solve the optimization problem in (4).

A. Problem decomposition

Due to the large dimension of action space, it will take a significant amount of time to achieve convergence. Moreover, the direct application of RL for solving the optimization problem in (4) becomes impractical with the growth of cache size, number of users, and number of files. To address the issue, we decompose the joint content cache and delivery problem into two subproblems, as follows.

P1 (Coordinated proactive cache updating): The subproblem P1 determines the proactive cache updating decisions according to the following foresighted optimization problem over a time horizon $t \in \mathcal{T}$

$$\min_{\mathbf{a}_m^t} \sum_{t\in\mathcal{T}} \sum_{f\in\mathcal{F}} P_{m,f}^t a_{m,f}^t, \quad \text{s. t. (5).} \quad (8)$$

Since the optimal solution to the problem in (8) requires look-ahead knowledge of the caching parameters, the assumption of obtaining them instantaneously violates the causality principle. Hence, the myopic optimization per time-slot would compromise the optimality in the long run due to the coupling amongst the optimization variables across time. Furthermore, the decision variables $\mathbf{a}_m^t$ should be determined coordinately among the distributed servers in order to minimize the overall system cost.

P2 (Local request processing): Subproblem P2 determines a source for server m, either another server in the neighborhood or the cloud, from which a missing content can be fetched cost-efficiently, by making a decision on $\mathbf{h}_m^t$ according to the following optimization problem,

$$\min_{\mathbf{h}_m^t} \sum_{f\in\mathcal{F}} P_{m,m',f}^t h_{m,m',f}^t + P_{m,c,f}^t h_{m,c,f}^t, \quad \text{s. t. (6).} \quad (9)$$

Remark 1: At the beginning of time slot t, edge server m receives user requests $\{D_{m,u}^t\}_{u\in\mathcal{U}_m}$. The requests are first served by the local cache at the m-th server, if those files are locally stored. Otherwise, the edge server initiates a content migration action to retrieve the requested files. The content retrieving occurs during the coordinated proactive caching iterations in each time slot, which requires reactive decisions to initiate content migration actions $\mathbf{h}_m^t$ within that time slot.

The policy governing the concurrent content migrations across distributed edge servers, according to *Remark 1*, requires a coordinated adjustment and maintenance of all servers' cache statuses efficiently, which is a substantial task due to the distributed nature of the problem. To address this issue, we develop a federated DQL caching algorithm, detailed in subsection IV-B, to optimize cache update policies at distributed edge servers according to subproblem P1, (8)). With the cache update policy determined for each server, we employ a greedy algorithm to find an optimal $\mathbf{h}_m^{t*}$ as a solution to subproblem (9), discussed in subsection IV-C. Finally, we analyze the computational complexity of the proposed unified federated DQL caching scheme in subsection IV-D.

B. Federated DQL Caching Algorithm

In this subsection, we introduce a federated DQL caching algorithm composed of a scalable DQL approach, to learn the heterogeneous users' content demands and the FL [10] to coordinate among multiple servers by aggregating their local DQL model parameters. The proposed federated DQL algorithm involves three functional stages. In the first stage, each edge server trains the local DQL model according to its local observations. In the second stage, the global server, which is deployed in the cloud center, chooses a number of edge servers to acquire their local model parameters for training the global DQL model. In the third stage, the trained global model parameters are distributed to the edge servers for further refining their local DQL models. Thereafter, each

edge server makes caching decisions based on the globally updated local model. In the following, we first introduce the proposed scalable DQL algorithm for each edge server. Next, the FL-empowered training process for the global DQL model is presented.

1) Proposed scalable DQL algorithm: Considering the simplicity and effectiveness of DQL, we develop a novel scalable algorithm based on the conventional DQL. Notably, the dimension of the search space, i.e., the actions $\mathbf{a}_m^t = \{a_{m,f}^t\}_{f\in\mathcal{F}}$, for an optimal solution to the problem in (8) is 2^F, which indicates an exponential growth with the number of contents. Thus, adopting the conventional DQL structure wherein each neuron of the output layer of the network represents a potential action [9], would require a highly complex deep neural network for fitting and the training process would be time-consuming and difficult to converge.

To break the curse of dimensionality and to prepare for the scalability of the network, we redesign the output layer of the standard DQL so that each neuron therein represents the *soft Q value* of a specific file. In this way, the number of output neurons of DQL is significantly decreased from 2^F to F. In the proposed DQL structure which is illustrated in Fig. 2, an output neuron indicates the *predicted importance* of a specific file and hence, a larger soft Q value of a file indicates its higher popularity in terms of being more likely to be requested by the users. The proposed restructured DQL model prepares for the scalability of the edge caching network by allowing for a significantly greater size of the content library.

Let the proposed outputs of the m-th server's DQL in Fig. 2 at time slot t be denoted by $\mathbf{Q}_m(\mathbf{s}_m^t;\boldsymbol{\theta}_m)$, and expressed as

$$\mathbf{Q}_m(\mathbf{s}_m^t;\boldsymbol{\theta}_m) = \left[Q_{m,1}(\mathbf{s}_m^t;\boldsymbol{\theta}_m),\ldots,Q_{m,F}(\mathbf{s}_m^t;\boldsymbol{\theta}_m)\right], \quad (10)$$

where the weights of the DQL network are stacked in vector $\boldsymbol{\theta}_m$ and element $Q_{m,f}(\mathbf{s}_m^t;\boldsymbol{\theta}_m), \forall f\in\mathcal{F}$ represents the corresponding soft Q value of file f at time slot t.

Since, in contrast to the conventional DQL network, each output neuron in the proposed DQL no longer indicates a potential action, it is infeasible to directly choose an action (i.e., a neuron) from the output layer of the network. Next, we introduce a two-phase procedure for the action selection, namely, Phase I: value updating phase; and Phase II: decision making phase, as follows.

Phase I (Value Updating Phase): The soft Q value of a content file f at a time slot t is updated according to

$$Q_{m,f}(\mathbf{s}_m^t;\boldsymbol{\theta}_m) \leftarrow (1-\beta)Q_{m,f}(\mathbf{s}_m^t;\boldsymbol{\theta}_m) + \beta Q_{m,f}(\mathbf{s}_m^{t-1};\boldsymbol{\theta}_m), \quad (11)$$

where $\beta\in(0,1)$. This value updating operation is applied to smooth out potentially large fluctuations due to the variations of state $\mathbf{s}_m$ at the input of the proposed DQL network.

Phase II (Decision Making Phase): In RL, the ϵ-greedy algorithm is commonly used for action selection process in order to balance exploitation and exploration. In the sequel, we adopt a similar ϵ-greedy idea for action selection, while with an improvement on the exploration, i.e., request-driven exploration.

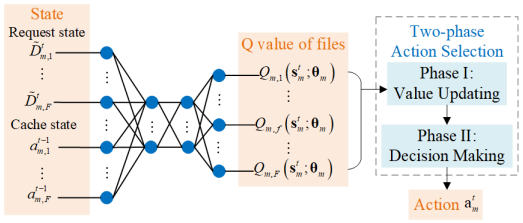

Fig. 2. The proposed architecture of deep neural network, where each output neuron represents the soft Q value of a file.

Exploitation: The greedy action for exploitation is to select the files with the highest updated soft Q values. Specifically, we sort all files by their corresponding soft Q values in a descending order. Let the first N_m files with largest Q values at time slot t denote by $\mathcal{F}_{\text{cache}}^t = \{\tilde{f}_1,\tilde{f}_2,\ldots,\tilde{f}_{N_m}\}$. Then the greedy action for the server m at time slot t is to cache the files in $\mathcal{F}_{\text{cache}}^t$, i.e., $\mathbf{a}_{max}^t = \{a_{m,f}^t = 1 | f\in\mathcal{F}_{\text{cache}}^t, a_{m,f}^t = 0 | f\in\mathcal{F}-\mathcal{F}_{\text{cache}}^t\}$.

Request-driven Exploration: In the classical DQL, exploration is usually performed by selecting the non-greedy actions with an equal probability, which greatly limits the effectiveness of exploration. Consequently, we propose an exploration approach based on accumulated requesting times of files, which are updated by the historical request information stored in states. We denote the accumulated requesting times of all files stored at server m by $\mathbf{v}_m^t = \{v_{m,f}^t = \sum_{i=0}^t \tilde{D}_{m,f}^i | f\in\mathcal{F}\}$. All elements in $\mathbf{v}_m^t$ are sorted in a descending order, and then the corresponding first N_m files with the highest values are collected as a set and denoted by $\mathcal{F}_{\mathbf{v}_m}^t$. Accordingly, the non-greedy action (exploration) $\mathbf{a}_{\mathbf{v}_m}^t$ for the server m at time slot t is to cache the files in $\mathcal{F}_{\mathbf{v}_m}^t$, i.e., $\mathbf{a}_{\mathbf{v}_m}^t = \{a_{m,f}^t = 1 | f\in\mathcal{F}_{\mathbf{v}_m}^t, a_{m,f}^t = 0 | f\in\mathcal{F}-\mathcal{F}_{\mathbf{v}_m}^t\}$.

In a word, at each time slot t, the agent m selects the action according to the following policy with a likelihood of $1-\epsilon$ or ϵ, i.e.,

$$\mathbf{a}_m^t = \begin{cases} \mathbf{a}_{max}^t, & \text{with } 1-\epsilon, \\ \mathbf{a}_{\mathbf{v}_m}^t, & \text{with } \epsilon. \end{cases} \quad (12)$$

In addition to the two-phase action selection procedure introduced above, the proposed DQL algorithm contains other three key modules: evaluation neural network, target neural network and experience replay buffer.

The evaluation neural network with parameters stacked in $\boldsymbol{\theta}_m$ are trained to approximate the soft Q values of files, while the target neural network with parameters $\hat{\boldsymbol{\theta}}_m$ is used for obtaining the target Q values. Both of them are constructed with the same architecture, where the input of the network is the state defined in subsection III-C, and the outputs are designed as the soft Q values of the files. The experience replay buffer $\mathcal{D}_m$ is introduced to store the server's past experiences for further model training.

The target Q values are updated according to

$$\mathbf{y}_m = \mathbf{r}_m^t + \gamma\mathbf{Q}_m(\mathbf{s}_m^{t+1};\hat{\boldsymbol{\theta}}_m), \quad (13)$$

where $\gamma\in[0,1]$ is the discount factor, the vector $\mathbf{r}_m^t = [r_{m,1}^t, r_{m,2}^t, ..., r_{m,F}^t]$ contains the reward values of each content file at time slot t as its F dimensions, and $\mathbf{Q}_m(\mathbf{s}_m^{t+1};\hat{\boldsymbol{\theta}}_m)$

is the output of the target neural network. Let the temporal-difference (TD) error as a function of $\boldsymbol{\theta}_m$ be defined as

$$\delta(\boldsymbol{\theta}_m) = \left[\mathbf{y}_m - \mathbf{Q}_m(\mathbf{s}_m^t; \boldsymbol{\theta}_m)\right] \odot \mathbf{a}_m^t, \tag{14}$$

where $\odot$ denotes element-wise multiplication. Note that only cached files at time slot t have non-zero values according to (14). Then, the evaluation neural network parameters $\boldsymbol{\theta}_m$ are updated via the stochastic gradient descent iterations

$$\boldsymbol{\theta}_m^{t+1} = \boldsymbol{\theta}_m^t - \alpha \nabla_{\boldsymbol{\theta}_m} L(\boldsymbol{\theta}_m), \tag{15}$$

where $\alpha \in [0, 1]$ is the learning rate and the loss function $L(\boldsymbol{\theta}_m)$ is defined by the squared $\mathcal{L}_2$-norm of $\delta(\boldsymbol{\theta}_m)$ as

$$L(\boldsymbol{\theta}_m) = \left\|\delta(\boldsymbol{\theta}_m)\right\|_2^2. \tag{16}$$

Finally the target neural network parameters $\hat{\boldsymbol{\theta}}_m$ are periodically synchronized with the evaluation network parameters $\boldsymbol{\theta}_m$ per K time slots. Since the target neural network is only used for obtaining the target Q value in (13), we will focus on the evaluation neural network (i.e., the trained DQL models) in all the following sections.

2) Smart agent selection: To further improve the performance of distributed DQL models trained by edge servers, we next propose a federated DQL caching algorithm that achieves globally coordinated DQL models at individual servers. The proposed algorithm significantly alleviates the excessive signaling overhead required by collecting all the experiences from the edge severs in a centralized unit by developing a RL algorithm based on TS [23]. Aligned with the general FL architecture [24], we develop an FL-empowered training process for global coordination of DQL models in two phases, namely: 1) agent selection; 2) model aggregation and distribution. In the following, we first introduce the proposed smart agent selection method.

In collaborative caching system, edge servers repeatedly perform cache updating while interacting with the environment. Naturally, there exists some edge servers achieving better caching performance compared to the others. Therefore, smart selection of participating edge servers in FL is central for better performance and reduced network complexity.

Let $L(\boldsymbol{\theta}_G)$ define the global loss function as

$$L(\boldsymbol{\theta}_G) = \sum_{m \in \mathcal{M}} \frac{|\mathcal{D}_m|}{D} L(\boldsymbol{\theta}_m), \tag{17}$$

where $\mathcal{D}_m, \forall m \in \mathcal{M}$ is the local dataset of server m, and $D = \sum_{m \in \mathcal{M}} |\mathcal{D}_m|$. The aim of the proposed smart agent selection for FL is to minimize the global loss function (17) in the long run, which is equivalent to the minimization of TD errors of all servers, defined in (14). The TD error is the difference between the target Q value and the approximated Q value, and the target Q value is actually the expected reward. Thus, the minimization of TD errors of all servers can be regarded as the maximization of the long-term accumulated rewards of all servers. Consequently, the agent selection problem can be formulated as a long-term reward maximization problem, i.e.,

$$\max_{\mathbf{x}^t} \sum_{t \in \mathcal{T}} \sum_{m \in \mathcal{M}} x_m^t \cdot r_m^t, \tag{18}$$

where $\mathbf{x}^t = \{x_1^t, x_2^t, ..., x_M^t\}$, and each element $x_m^t \in \{0, 1\}$ is a binary variable to indicate whether server m is selected at time slot t. Server m is selected at time slot t if $x_m^t = 1$, and $x_m^t = 0$, otherwise. Let $\mathcal{A}_x$ denote the set of all possible solutions for agent selection problem in (18). Hence, the number of possible solutions for $\mathbf{x}^t$ is $|\mathcal{A}_x| = 2^M$.

To avoid biased selection of servers in which only the servers with current best performance are chosen, we propose a smart agent selection method based on TS. TS is a RL algorithm for online decision making, where actions are taken sequentially to better balance exploitation and exploration, so as to maximize the achievable accumulated rewards.

Supposing that the global server performs agent selection every T_s time slots, we calculate the sum of accumulated rewards obtained by all servers in T_s consecutive time slots as

$$R_{\mathbf{x}^t} = \sum_{i=t-T_s+1}^{t} \sum_{m \in \mathcal{M}} r_m^i. \tag{19}$$

Furthermore, we normalize $R_{\mathbf{x}^t}$ as $\hat{R}_{\mathbf{x}^t} \in [0, 1]$ and interpret the normalized reward $\hat{R}_{\mathbf{x}^t}$ as the success probability of selecting action $\mathbf{x}^t$, which will be used in the sequential operations of TS.

In TS algorithm, the Beta distribution with parameters $a_{\mathbf{x}^t}$ and $b_{\mathbf{x}^t}$ is generated for each action $\mathbf{x}^t \in \mathcal{A}_x$, to estimate the probability of selecting the action $\mathbf{x}^t$, which is denoted as $\eta_{\mathbf{x}^t} \in [0, 1]$. Recall that the Beta distribution is a probability distribution on probabilities, hence, its domain is bounded between 0 and 1. Namely, $\eta_{\mathbf{x}^t}$ is a probability random variable drawn from the Beta distribution, i.e., $\eta_{\mathbf{x}^t} \sim \text{Beta}(a_{\mathbf{x}^t}, b_{\mathbf{x}^t})$. Once the action $\mathbf{x}^t$ is selected, the parameters of corresponding Beta distribution will be updated according to the following rule

$$\begin{cases} a_{\mathbf{x}^{t+1}} \leftarrow a_{\mathbf{x}^t} + u_{\mathbf{x}^t} \\ b_{\mathbf{x}^{t+1}} \leftarrow a_{\mathbf{x}^t} + 1 - u_{\mathbf{x}^t}, \end{cases} \tag{20}$$

where $u_{\mathbf{x}^t}$ is an independent sample drawn from the Bernoulli distribution with success probability $\hat{R}_{\mathbf{x}^t}$. If $\hat{R}_{\mathbf{x}^t}$ is close to 1, which implies that the action $\mathbf{x}^t$ could potentially improve the overall caching performance, there is a high probability that $a_{\mathbf{x}^{t+1}}$ increases by 1 while $b_{\mathbf{x}^{t+1}}$ keeps unchanged, and the mean of Beta distribution increases accordingly. On the contrary, if $\hat{R}_{\mathbf{x}^t}$ is close to 0, which implies that the action $\mathbf{x}^t$ may not benefit the overall caching performance, there will be a high probability that $b_{\mathbf{x}^{t+1}}$ increases by 1 while $a_{\mathbf{x}^{t+1}}$ keeps unchanged, and the mean of Beta distribution decreases accordingly.

Note that only the parameters of a selected action are updated via (20), and the proposed smart agent selection is performed by the global server according to

$$\mathbf{x}^t = \arg \max_{\mathbf{x}' \in \mathcal{A}_x} \eta_{\mathbf{x}'}. \tag{21}$$

3) Model aggregation and distribution: Different from the existing FL algorithms, e.g., FedAvg algorithm [10], which aggregates all the local model parameters to the global model, we preserve the network output layer locally. This is because,

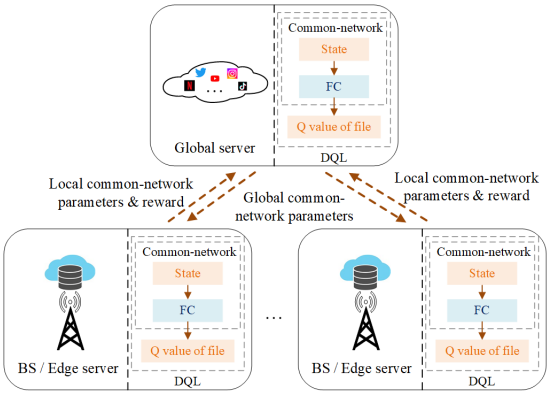

Fig. 3. The diagram of proposed federated DQL caching algorithm.

the network output layer in our proposed framework is designed to represent the *predicted importance* of files, which can be regarded as the specific user preferences of each file at the individual server. Therefore the parameters of the output layer should be trained individually by each server due to the request heterogeneity of covered users.

The early layers of neural network are referred to as the common-network part, and we denote $\boldsymbol{\theta}_m^c, \boldsymbol{\theta}_m^o, m \in \mathcal{M}$, respectively, as the common-network parameters and the parameters of the output layer. As illustrated in Fig.3, in each training round of FL, the selected servers only upload their parameters of the common-network part to the global server. Then the global common-network parameters $\boldsymbol{\theta}_G^{c\,t}$ are updated according to the following rule

$$\boldsymbol{\theta}_G^{c\,t} = \frac{1}{\sum_{m\in\mathcal{M}_G^t}|\mathcal{D}_m|}\sum_{m\in\mathcal{M}_G^t}|\mathcal{D}_m|\boldsymbol{\theta}_m^{c\,t}, \tag{22}$$

where $\mathcal{M}_G^t$ is the set of selected agents at time slot t.

After aggregation, the updated global parameters $\boldsymbol{\theta}_G^{c\,t}$ are distributed to all edge servers. Next, each server combines the newly obtained common-network parameters with the locally trained output layer parameters to form the updated parameters of its local DQL model, i.e., $\boldsymbol{\theta}_m^{t+1} \leftarrow \{\boldsymbol{\theta}_G^{c\,t}, \boldsymbol{\theta}_m^{o\,t}\}$. Based on the newly updated local model, each server makes caching decisions while performing local model training in the subsequent time slots.

C. Local request processing

As discussed in subsection IV-A, the content migration decisions are reactively made during each time slot when edge servers encounter cache misses. Taking into account that the number of user requests received at each time slot is finite and the edge servers aim to fetch files with the lowest cost, we introduce a greedy algorithm to find the optimal content migration action and solve the problem in (9), as described below.

It is noted that the content migration action comprises the fetching action $\left\{h_t^{m,m',f}\middle|m' \in \mathcal{M}^m, f \in \mathcal{F}\right\}$ and the retrieving action $\left\{h_t^{m,c,f}\middle|f \in \mathcal{F}\right\}$. The retrieving action makes sense and can only be determined after the fetching action is determined. In our file fetching protocol, the local server

Algorithm 1 The Proposed Unified Federated DQL Caching Scheme

Initialize: $\{\boldsymbol{\theta}_m, \hat{\boldsymbol{\theta}}_m, \mathcal{D}_m\}_{m\in\mathcal{M}}$. The global model server in cloud center is initialized.
for $t \in \mathcal{T}$ **do**
  for Each edge server $m \in \mathcal{M}$ **in parallel do**
    Receive content requests from associated users and observe the current state $\mathbf{s}_m^t$.
    Take cache updating action $\mathbf{a}_m^t$ according to two-phase action selection in (12).
    Get content migration action $\mathbf{h}_m^t$ according to (23) and (24).
    Compute the reward of each file $r_{m,f}^t$ to form the vector of file rewards $\mathbf{r}_m^t$, and observe the next state $\mathbf{s}_m^{t+1}$.
    Store the transition $\{\mathbf{s}_m^t, \mathbf{a}_m^t, \mathbf{r}_m^t, \mathbf{s}_m^{t+1}\}$ into the experience replay buffer $\mathcal{D}_m$.
    Sample a mini-batch of experiences from the buffer $\mathcal{D}_m$ to update the evaluation neural network parameters $\boldsymbol{\theta}_m^t$ by minimizing the loss function defined in (16) using stochastic gradient descent algorithm.
    if $t \mod K = 0$ **then**
      Replace the parameters of target neural network $\hat{\boldsymbol{\theta}}_m^t \leftarrow \boldsymbol{\theta}_m^t$.
    end if
  end for
  if $(t \mod T_s) = 0$ **then**
    The global server selects a set of servers $\mathcal{M}_G^t$ according to the proposed smart agent selection method in (21).
    Each selected server uploads its common-network parameters $\boldsymbol{\theta}_m^{c\,t}, m \in \mathcal{M}_G^t$ to the global server.
    The global server aggregates global parameters $\boldsymbol{\theta}_G^{c\,t}$ based on (22).
    The global server send $\boldsymbol{\theta}_G^{c\,t}$ back to all edge servers.
    Each server $m \in \mathcal{M}$ updates its local model with newly obtained parameters as $\boldsymbol{\theta}_m^{t+1} \leftarrow \{\boldsymbol{\theta}_G^{c\,t}, \boldsymbol{\theta}_m^{o\,t}\}$.
  end if
end for

consistently fetches the file f with the lowest fetching cost. Thus, the optimal fetching action is given by

$$h_{m,m',f}^{t*} = \begin{cases} 1, \text{ if } \prod_{j=1}^{i-1}(1-a_{(j)_m,f}^t)a_{(i)_m,f}^t = 1, \\ 0, \text{ otherwise }, \end{cases} \tag{23}$$
$$\forall m' \neq m, m \in \mathcal{M}, m' \in \mathcal{M}_m, f \in \mathcal{F}, t \in \mathcal{T},$$

where $m' = (i)_m$ is the server with i-th lowest fetching cost in terms of server m.

Once the optimal fetching action have been decided, the optimal retrieving action can be sequentially calculated as

$$h_{m,c,f}^{t*} = \begin{cases} 1, \text{ if } a_{m,f}^t + \sum_{m'\in\mathcal{M}} h_{m,m',f}^t = 0, \\ 0, \text{ otherwise }, \end{cases} \tag{24}$$
$$m \in \mathcal{M}, f \in \mathcal{F}, t \in \mathcal{T}.$$

In a nutshell, the proposed unified federated DQL caching scheme is summarized in Algorithm 1.

D. Complexity analysis

In this paper, we adopt the fully connected (FC) networks to construct the DQL models. Let Z and n_z denote the number of hidden layers of the DQL model, and the number of neurons in the z-th hidden layer, respectively. The time complexity of the adopted DQL can be expressed as $\Phi = J(2F \cdot n_1 + \sum_{z=1}^{Z-1} n_z \cdot n_{z+1} + F \cdot n_Z)$, where $J(\cdot)$ is the time complexity of updating the parameters of fully connected layers. Thus, the complexity of backpropagation for training a DQL model in a single transition is $\mathcal{O}(\Phi)$. Since the local DQL models are trained in parallel, we only consider the complexity of one local model. Therefore, the time complexity of the proposed framework is $\mathcal{O}(\Phi \times N \times T)$, where N is the number of transitions sampled in each training round.

V. Performance Evaluation

In this section, we set up simulation scenarios to evaluate the performance of our developed caching scheme under various settings and compare the results against some of the well-known existing caching approaches.

A. Simulation settings and baselines

We use the same user request model as in [25], where the transition probability $p_{i,j}$ is given by

$$p_{i,j} = \begin{cases} P_0, \ i \in \bar{\mathcal{F}}, j = 0, \\ (1-P_0)\frac{\frac{1}{j^\lambda}}{\sum_{j'=1}^{F}\frac{1}{j'^\lambda}}, \ i = 0, j \in \mathcal{F}, \\ (1-P_0)\frac{1}{G}, \ i \in \mathcal{F}, j = (i+g)\text{mod}(F+1), \\ \qquad\qquad g \in \{1,2,\ldots,G\}, \\ 0, \ \text{otherwise}. \end{cases} \tag{25}$$

Note that the user request model is parameterized by $\{P_0, \lambda, G\}$, where P_0 is the transition probability to no request by a user from its any current request state. For $i = 0$, i.e., the user does not have a request at current time slot, the probability of requesting any file $f \in \mathcal{F}$ follows a Zipf-like distribution with parameter λ. Let us assign a set of G neighbouring files to each file $i \in \mathcal{F}$, i.e., $\mathcal{G}_i = \{f \in \mathcal{F} : (i+g)\text{mod}(F+1), g \in \{1,2,\ldots,G\}\}$. Then, the transition probability from any file $i \in \mathcal{F}$ to a neighbouring file $j \in \mathcal{G}_i$ follows a uniform distribution, and the transition probability from a file $i \in \mathcal{F}$ to a non-neighbouring file $j \notin \mathcal{G}_i$ is set to zero.

We assign $n_f = 1$ for all $f \in \mathcal{F}$. The unit caching cost and the unit retrieving cost are set to be $\tau^{\text{cache}} = 1, \tau^{\text{ret}} = 20$, respectively. Various experiments involve adjusting parameters such as content library size F, the number of edge servers M, the number of users U_m, user request model parameters $\{P_0, \lambda, G\}$, and the unit fetching cost $\tau^{\text{fet}}_{mm'}$. The specific values for these parameters will be explicitly stated in the corresponding experiment contexts.

TABLE I
Parameter settings

Parameter	Value	Parameter	Value
Discount factor (γ)	0.9	Common-network part	$2F \times 1024 \times 512$
Replay buffer size	2000	Output layer	$512 \times F$
Mini-batch size	128	Optimizer	Adam
Q value discount rate (β)	0.95	Learning rate (α)	0.01
Agent selection frequency (T_s)	100	Target network updating frequency (K)	50

We construct the neural network for each edge server with an identical architecture, as follows: one input layer, two hidden layers with ReLu activation and one output layer with sigmoid activation. Unless specified otherwise, other main simulation parameters are given in Table I. Note that we employ the above parameter settings in the simulation just as an illustration to evaluate our proposed caching algorithm, and the choice of parameters will not affect the effectiveness of our proposed algorithm.

We compare the performance of the proposed algorithm against four baseline algorithms, namely: 1) Hierarchical reinforcement learning (HRL) based cache scheme proposed in [18], which adopts classical DQN to optimize the caching policies at BSs. Particularly in [18], the number of neural network outputs, representing the dimensionality of the action space, is chosen as the combination number of N_m (cache capacity of server m) in F (total number of files), denoted as $\binom{F}{N_m}$. Furthermore in [18], the caching policy implemented on the BS side is designed for each individual server, and it does not involve exploration of inter-server cooperation. 2) Distributed training scheme, where each server has the same neural network architecture as our proposed DQL model. However, each server trains its own model using local experiences independently. There is no global server who offers information exchange among edge servers, and all servers make caching decisions based on their individually trained models. Alternatively, we can refer to this scheme as the proposed method without FL. 3) LFU [12], which replaces one cached file having the lowest frequency of request with a new request; 4) LRU [11], which replaces the cached file having the least frequency of recent requests with a new request.

B. Experiments on small-scale scenarios

In this subsection, we compare the performance of the proposed algorithm with HRL in [18] under relatively small-scale scenarios with $M = 2$ edge servers, where each server covers $U_1 = U_2 = 10$ users. We assume that the request model for users covered by server 1 is parameterized by $\{P_0 = 0.2, \lambda = 0.8, G = 5\}$, and the request model for users covered by server 2 is parameterized by $\{P_0 = 0.1, \lambda = 0.6, G = 3\}$. The unit fetching cost $\tau^{\text{fet}}_{12} = 5$.

For the sake of fairness and clarity, we implement the HRL algorithm following the design in [18], where the number of output neurons is determined by the combination number of N^m in F (i.e., $\binom{F}{N_m}$). We maintain the consistency of other neural network parameters and training hyperparameters of HRL with our settings presented in subsection V-A.

We use cache hit rate as a performance metric, wherein the count of cache hits is calculated as the number of requests fulfilled by any of the edge servers within the system. Hence,

TABLE II
COMPARISON OF THE PROPOSED ALGORITHM AND THE BASELINE HRL ALGORITHM UNDER VARIOUS CONTENT LIBRARY SIZES AND DIFFERENT NUMBER OF SERVER CACHE STORAGE. (THE EXACT NUMBER OF OUTPUT NEURONS FOR THE HRL ALGORITHM IS $\binom{F}{N_m}$, WHILE APPROXIMATE VALUES ARE PROVIDED IN THIS TABLE FOR BETTER COMPARISON.)

Algorithm	F	N_m	#Output neurons	Cache hit rate	Runtime/s
Proposed	20	5	20	46.78%	7.92
HRL	20	5	$20 \times 7.8 \times 10^2$	29.02%	6.51
Proposed	40	5	40	22.50%	8.68
HRL	40	5	$40 \times 1.6 \times 10^4$	16.06%	104.52
Proposed	60	5	60	15.25%	10.04
HRL	60	5	$60 \times 9.1 \times 10^4$	11.88%	1117.13
Proposed	20	10	20	74.48%	24.01
HRL	20	10	$20 \times 9.2 \times 10^3$	54.46%	42.19
Proposed	40	10	40	44.83%	21.79
HRL	40	10	$40 \times 2.1 \times 10^7$	–	–
Proposed	60	10	60	31.78%	26.88
HRL	60	10	$60 \times 1.3 \times 10^9$	–	–

the cache hit rate is defined as the proportion of the total cache hits to the overall number of content requests. The runtime of each algorithm is used as another performance criterion.

We evaluate the proposed algorithm against the baseline HRL in various scenarios involving different sizes of content library and cache storage per server. Specifically, the content library size F ranges from 20 to 60, and the cache storage for each server is set to be 5 and 10, as presented in Table II. Note that in Table II, the number of output neurons for HRL is computed as approximate (but close to accurate) values for the ease of comparison. For example, if $F = 20, N_m = 5$, the exact number of output neurons for HRL is calculated as the combination number $\binom{20}{5} = 15504$. Whereas, we approximate the number of output neurons for HRL based on multiples of the content library size $F = 20$, resulting in the rounded value of $20 \times 7.8 \times 10^2$.

As depicted in Table II, the cache hit rate of the proposed algorithm surpasses that of HRL across various settings, as the former makes full use of cooperation among the edge servers. From a runtime perspective, Table II demonstrates a rapid increase in the runtime of the HRL algorithm as the size of the content library and server storage grows. Notably, HRL fails to accomplish the task when F and N_m keep increasing, as a result of the sharply growing number of output neurons in the neural networks. In contrast, our proposed algorithm, with an output layer consisting of only F neurons, remains robust. A surge in the size of the content library and server storage does not lead to significant performance degradation for our proposed algorithm. The robustness of our proposed algorithm is further verified by experiments conducted in the following subsections, where scenarios with a larger content library are considered. Since HRL cannot be executed normally under scenarios with larger sizes of content library and cache storage, we will not consider it as a baseline in the subsequent experiments.

C. Experiments with the number of servers $M = 6$

In this subsection, we conduct experiments under a larger-scale scenario with $M = 6$ edge servers and $F = 1000$ files. Each server covers different number of users, i.e., $U_1 = 20, U_2 = 30, U_3 = 40, U_4 = 50, U_5 = 60, U_6 = 70$. We assume each server has $|\mathcal{M}_m| = 4$ proximal servers, and the unit fetching costs are set as $\tau_{12}^{\text{fet}} = \tau_{23}^{\text{fet}} = 3, \tau_{34}^{\text{fet}} = \tau_{45}^{\text{fet}} = 4, \tau_{56}^{\text{fet}} = \tau_{61}^{\text{fet}} = 5, \tau_{13}^{\text{fet}} = 6, \tau_{24}^{\text{fet}} = 7, \tau_{26}^{\text{fet}} = \tau_{35}^{\text{fet}} = 8, \tau_{46}^{\text{fet}} = 9, \tau_{15}^{\text{fet}} = 10$. Additionally, we consider time-variable request models for different sets of users to achieve heterogeneous user requests, i.e., diverse user demand scenarios with dynamic file popularity. The parameters of user request model are aperiodically and randomly drawn from $P_0 \in \{0.1, 0.15, 0.2, 0.3\}, \lambda \in \{0.8, 1.0, 1.1, 1.2\}, G \in \{25, 40, 50, 75\}$.

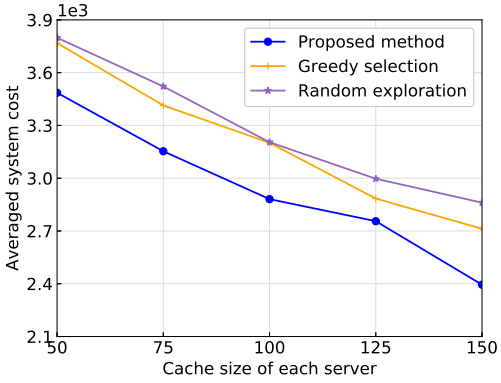

Fig. 4. Averaged system cost with different exploration methods.

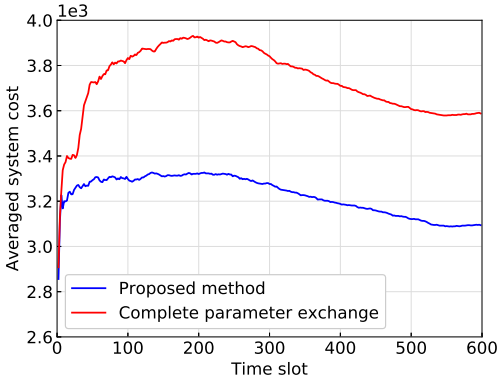

Fig. 5. Averaged system cost with different DQL model parameter exchange methods with $N_m = 100, \forall m \in \mathcal{M}$.

1) Performance with different exploration methods: In Fig. 4, we evaluate the performance of the proposed algorithm using proposed request-driven exploration, random exploration (i.e., each non-greedy action is selected with a uniform probability for exploration), and a greedy action selection scheme (i.e., only the files with the highest soft Q values are cached). As can be seen from Fig. 4, the proposed request-driven exploration method achieves the lowest system cost compared to other two schemes across various cache sizes. This performance gain attributes to exploiting the historical request information, which enables each server to cache files with higher popularity, rather than blind exploration.

2) Performance with different DQL model parameter exchange methods: As mentioned in Section IV-B3, we train the output layer parameters of each server individually to preserve

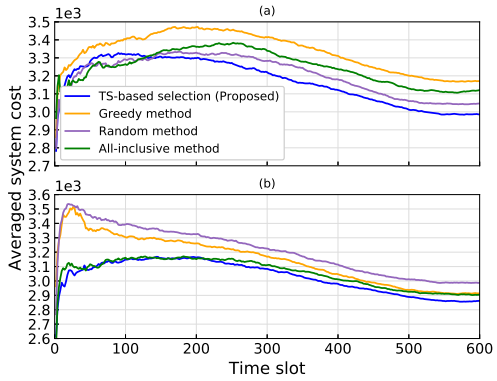

Fig. 6. Averaged system cost with various agent selection methods under different parameter configurations. (a) Identical cache size for each server: $N_m = 100, \forall m \in \mathcal{M}$. (b) Different cache sizes for edge servers: $N_1 = 60, N_2 = 80, N_3 = 100, N_4 = 120, N_5 = 140, N_6 = 160$.

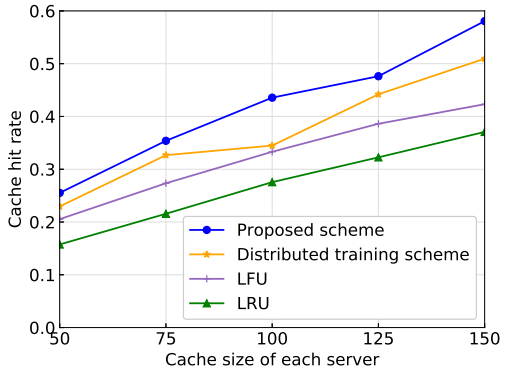

Fig. 7. Comparison of cache hit rate of the proposed algorithm, distributed training scheme, LFU and LRU with different cache sizes.

the user preferences at different servers, instead of aggregating all the local model parameters to the global model. To justify this design, we investigate the system cost of the proposed algorithm with different DQL model parameter exchange methods in Fig. 5. It is observed that the overall system cost of the proposed method is significantly lower than that of the complete parameter exchange method over time. This improvement is attributed to the fact that the proposed scheme learns the cache updating policies more effectively, benefiting from well-preserved local user preferences and more efficient global coordination among the DQN models at distributed edge servers.

3) Performance with different agent selection methods: Next, we evaluate the performance of the proposed smart agent selection method with various parameter configurations, as illustrated in Fig. 6. Moreover, we compare the proposed smart agent selection method with greedy selection method, random selection method and all-inclusive method. In the greedy method, edge servers with currently highest accumulated rewards are selected to upload their local parameters for global model training. In the random method, edge servers are randomly selected for global model training. In the all-inclusive method, the local parameters of all edge servers are aggregated to update the global model parameters. Fig. 6(a) shows the averaged system cost of all four selection methods under the configuration where each server has an identical cache size, i.e., $N_m = 100, \forall m \in \mathcal{M}$. In contrast, the cache sizes of edge servers are set to be different as $N_1 = 60, N_2 = 80, N_3 = 100, N_4 = 120, N_5 = 140, N_6 = 160$ in Fig. 6(b). Since the aim of the proposed smart agent selection is to minimize the long-term system cost, it incurs slight performance losses due to exploration at the beginning, as shown in Fig. 6. Nevertheless, the overall performance of the proposed smart agent selection outperforms the other three methods. These results suggest the effectiveness of the proposed method under various parameter configurations.

TABLE III
PEAK MEMORY USAGE OF FOUR CACHING SCHEMES UNDER VARIOUS SETTINGS WITH $F = 1000$.

#Servers (M)	Cache size (N_m)	Proposed scheme/MB	Distributed training scheme/MB	LFU/MB	LRU/MB
6	50	728.85	728.86	156.67	156.67
	100	735.77	735.73	157.43	157.43
	150	742.49	742.52	158.20	158.20
12	50	1304.08	1304.04	247.07	247.07
	100	1316.22	1316.16	248.30	248.30
	150	1328.39	1328.36	249.53	249.53
24	50	2738.85	2728.83	490.14	490.14
	100	2885.77	2864.58	492.60	492.60
	150	2955.14	2928.18	495.07	495.07

4) Comparison of overall cache hit rate: As demonstrated in Fig. 7, we investigate the cache hit rate of proposed algorithm and three baselines, i.e., distributed training scheme, LFU and LRU, under different cache sizes. It shows that the cache hit rate of all schemes increases with the growing cache size of each server, which ranges from 50 to 150 units. Particularly, the proposed algorithm outperforms distributed training scheme, LFU and LRU across all parameter configurations. This improvement is mainly attributed to a more precise prediction of heterogeneous user requests, as well as more effective coordination among edge servers via the FL-empowered training process.

5) Comparison of peak memory usage: As shown in Table III, we compare the peak memory usage of four caching schemes under various settings. As the cache size and the number of servers increase, the peak memory usage for all caching schemes gradually rises. It is observed that our scheme and the distributed training scheme consume a similar amount of peak memory under the same setting. These peak memory usage values are approximately five times higher than those of the LFU/LRU schemes, as LFU/LRU do not involve any model training. Nonetheless, our proposed scheme surpasses the other three baselines in cache hit rate, as observed in Fig. 7 and Fig. 8. In practical scenarios, a high cache hit rate greatly enhances overall system performance by reducing latency and improving content access speeds, thus optimizing QoS and QoE. These substantial performance gains outweigh the costs associated with increased computational overhead.

D. Experiments with the number of servers $M = 25$

In this subsection, we investigate the robustness and scalability of our proposed caching scheme in an even larger-scale scenario involving $M = 25$ edge servers and $F = 1500$ files.

In contrast to the experiments conducted in subsection V-C, we adopt more flexible and varied parameter settings,

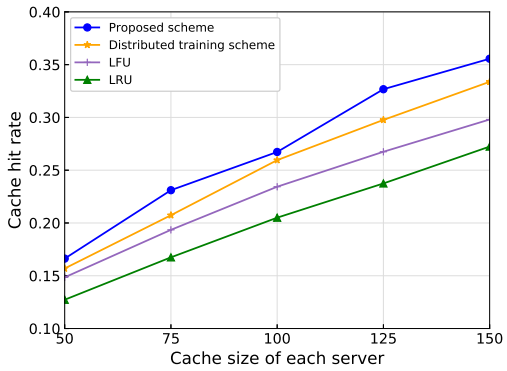

Fig. 8. Comparison of cache hit rate of the proposed algorithm, distributed training scheme, LFU and LRU under different cache sizes with $M = 25, F = 1500$.

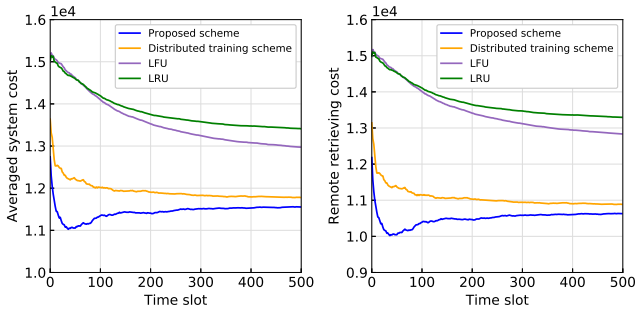

Fig. 9. Comparison of the proposed scheme, distributed training scheme, LFU and LRU in terms of averaged system cost and remote retrieving cost with $M = 25, F = 1500$. Besides, the cache sizes of each server are sampled at intervals of 5 from 80 to 200.

particularly regarding the number of users, user request models, as well as sets of proximal servers. Specifically, the number of users covered by each server is randomly drawn from $\{20, 30, 40, 50, 60\}$. The parameters of user request models are aperiodically and randomly drawn from $P_0 \in \{0.1, 0.15, 0.2, 0.25\}$, $\lambda \in \{0.9, 1.0, 1.1, 1.2\}$, and $G \in \{50, 75, 100, 125\}$. Each server is assumed to have a minimum of 2 proximal servers and a maximum of 4 proximal servers. The unit fetching costs are set as $\{\tau^{\text{fet}}_{mm'} = 3 | |m' - m| \leq 1, \tau^{\text{fet}}_{mm'} = 6 | 1 < |m' - m| \leq 2\}$.

1) Comparison of overall cache hit rate: As observed in Fig. 8, the proposed scheme outperforms the other three baselines across various cache sizes, even with a higher number of edge servers and content files, i.e., $M = 25$ and $F = 1500$. Particularly, the performance gap between the proposed scheme and distributed training scheme indicates the benefits of the FL-empowered training process. In other words, effective information exchange is achieved during the FL-empowered training process, and hence enhancing the caching performance at distributed edge servers.

2) Comparison of averaged system cost and remote retrieving cost: In the sequel, we compare the performance of four caching schemes in terms of averaged system cost and remote retrieving cost in the large-scale scenario with $M = 25, F = 1500$. Particularly, the cache sizes of each server are sampled at intervals of 5, ranging from 80 to 200. The results in Fig. 9 demonstrate the advantages of our proposed scheme over other three baselines in terms of both averaged system cost and remote retrieving cost. Moreover, these results in both Fig. 8 and Fig. 9 confirm the scalability and effectiveness of our proposed scheme.

VI. Conclusion

In this paper, we focused on the collaborative cache update and request delivery problem in an edge caching system with multiple edge servers and heterogeneous users. We first formulated the problem as a long-term system cost minimization problem. To learn the time-varying file popularity and capture the features of heterogeneous user requests, we restated the problem in terms of RL, thereby decomposing the problem into two subproblems in order to deal with the large dimensionality of the action space. A unified federated DQL caching scheme is therefore designed, consisting of a federated DQL algorithm and a greedy algorithm to tackle these two subproblems accordingly. Particularly, the proposed federated DQL caching algorithm leverages FL and the scalable DQL algorithm to train the distributed DQL models coordinately via the FL-empowered training process. Furthermore, a TS-based smart agent selection method is introduced to not only search for optimal server set to participate in the training process, but also reduce signaling overhead among edge servers and the global server. Additionally, to cope with the request heterogeneity and to preserve the users' preferences at each server, we decompose the DQL parameters, and the selected servers only exchange the common-network parameters with the global server. Numerical results for small-scale scenarios demonstrated that our proposed scheme showed a 29% average increase in cache hit rate and a 53% average decrease in runtime compared to the baseline HRL algorithm. Particularly, simulation results have confirmed the robustness and efficiency of our proposed scheme, when the content library size and the cache storage size increase. In large-scale scenarios, our proposed scheme demonstrated the lowest averaged system cost and the highest cache hit rate under various settings. Compared to distributed training scheme, LFU, and LRU, our proposed scheme achieved an average gain of 7%, 17% and 34%, respectively, in terms of cache hit rate, as well as an average decrease of 2%, 11% and 14%, respectively, in terms of averaged system cost.

References

[1] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[2] J. Zhang and K. B. Letaief, "Mobile edge intelligence and computing for the internet of vehicles," *Proceedings of the IEEE*, vol. 108, no. 2, pp. 246–261, 2020.

[3] ETSI. (2014, September) Mobile-edge computing-introductory echnical white paper. [Online]. Available: https://portal.etsi.org/Portals/0/TBpages/MEC/Docs/Mobile-edge_Computing_-_Introductory_Technical_White_Paper_V1\%2018-09-14.pdf

[4] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, "In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning," *IEEE Network*, vol. 33, no. 5, pp. 156–165, 2019.

[5] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[6] H. Wu, Y. Fan, Y. Wang, H. Ma, and L. Xing, "A comprehensive review on edge caching from the perspective of total process: Placement, policy and delivery," *Sensors*, vol. 21, no. 15, p. 5033, 2021.

[7] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[8] D. Gan, X. Ge, and Q. Li, "An optimal transport-based federated reinforcement learning approach for resource allocation in cloud–edge collaborative iot," *IEEE Internet of Things Journal*, vol. 11, no. 2, pp. 2407–2419, 2024.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.

[10] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.

[11] J. Wang, "A survey of web caching schemes for the internet," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 5, pp. 36–46, 1999.

[12] M. Bilal and S.-G. Kang, "A cache management scheme for efficient content eviction and replication in cache networks," *IEEE Access*, vol. 5, pp. 1692–1701, 2017.

[13] X. Zhang, G. Zheng, S. Lambotharan, M. R. Nakhai, and K. Wong, "A reinforcement learning-based user-assisted caching strategy for dynamic content library in small cell networks," *IEEE Transactions on Communications*, vol. 68, no. 6, pp. 3627–3639, 2020.

[14] Z. Sun and M. R. Nakhai, "Distributed learning-based cache replacement in collaborative edge networks," *IEEE Communications Letters*, vol. 25, no. 8, pp. 2669–2672, 2021.

[15] A. Sadeghi, G. Wang, and G. B. Giannakis, "Deep reinforcement learning for adaptive caching in hierarchical content delivery networks," *IEEE Transactions on Cognitive Communications and Networking*, vol. 5, no. 4, pp. 1024–1033, 2019.

[16] C. Zhong, M. C. Gursoy, and S. Velipasalar, "Deep multi-agent reinforcement learning based cooperative edge caching in wireless networks," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–6.

[17] D. Li, Y. Han, C. Wang, G. Shi, X. Wang, X. Li, and V. C. M. Leung, "Deep reinforcement learning for cooperative edge caching in future mobile networks," in *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, 2019, pp. 1–6.

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

[19] C. Zhong, M. C. Gursoy, and S. Velipasalar, "Deep reinforcement learning-based edge caching in wireless networks," *IEEE Transactions on Cognitive Communications and Networking*, vol. 6, no. 1, pp. 48–61, 2020.

[20] X. Wu, X. Li, J. Li, P. C. Ching, V. C. M. Leung, and H. V. Poor, "Caching transient content for iot sensing: Multi-agent soft actor-critic," *IEEE Transactions on Communications*, vol. 69, no. 9, pp. 5886–5901, 2021.

[21] C. Sun, X. Li, J. Wen, X. Wang, Z. Han, and V. C. M. Leung, "Federated deep reinforcement learning for recommendation-enabled edge caching in mobile edge-cloud computing networks," *IEEE Journal on Selected Areas in Communications*, vol. 41, no. 3, pp. 690–705, 2023.

[22] K. Psounis, A. Zhu, B. Prabhakar, and R. Motwani, "Modeling correlations in web traces and implications for designing replacement policies," *Computer Networks*, vol. 45, pp. 379–398, 07 2004.

[23] D. J. Russo, B. Van Roy, A. Kazerouni, I. Osband, Z. Wen *et al.*, "A tutorial on thompson sampling," *Foundations and Trends® in Machine Learning*, vol. 11, no. 1, pp. 1–96, 2018.

[24] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 2031–2063, 2020.

[25] Y. Sun, Y. Cui, and H. Liu, "Joint pushing and caching for bandwidth utilization maximization in wireless networks," *IEEE Transactions on Communications*, vol. 67, no. 1, pp. 391–404, 2019.

Ming Zhao (Student Member, IEEE) received her B.S. degree in communication engineering from Chongqing University, China, in 2016, and her MSc by research degree in electronic engineering from Univeristy of York, UK, in 2019. She is currently a Ph.D. student with the Department of Engineering, King's College London, UK. Her current interests include edge caching, reinforcement learning, and federated learning.

Mohammad Reza Nakhai (Senior Member, IEEE) received the B.Sc. and M.Sc. degrees in electrical engineering from the Sharif University of Technology, Tehran, Iran, in 1984 and 1987, respectively, and the Ph.D. degree in electronic engineering from King's College London, University of London, U.K., in 2000. From 1988 to 1995, he was with Sharif University of Technology as a member of Research Faculty. In 2001, he joined King's College London, where he is currently with the Department of Engineering. He served as an Editor for IEEE Transactions on Wireless Communications. His current research interests include machine learning and artificial intelligence for wireless communications applications, wireless network optimization for energy efficiency, cognitive radio communications and signal processing.