Multi-Agent Deep Reinforcement Learning Based Cooperative Edge Caching for Ultra-Dense Next-Generation Networks

Shuangwu Chen, Zhen Yao, Xiaofeng Jiang, Jian Yang, *Senior Member, IEEE*, and Lajos Hanzo, *Fellow, IEEE*

***Abstract*—The soaring mobile data traffic demands have spawned the innovative concept of mobile edge caching in ultra-dense next-generation networks, which mitigates their heavy traffic burden. We conceive cooperative content sharing between base stations (BSs) for improving the exploitation of the limited storage of a single edge cache. We formulate the cooperative caching problem as a partially observable Markov decision process (POMDP) based multi-agent decision problem, which jointly optimizes the costs of fetching contents from the local BS, from the nearby BSs and from the remote servers. To solve this problem, we devise a multi-agent actor-critic framework, where a communication module is introduced to extract and share the variability of the actions and observations of all BSs. To beneficially exploit the spatio-temporal differences of the content popularity, we harness a variational recurrent neural network (VRNN) for estimating the time-variant popularity distribution in each BS. Based on multi-agent deep reinforcement learning, we conceive a cooperative edge caching algorithm where the BSs operate cooperatively, since the distributed decision making of each agent depends on both the local and the global states. Our experiments conducted within a large scale cellular network having numerous BSs reveal that the proposed algorithm relying on the collaboration of BSs substantially improves the benefits of edge caches.**

***Index Terms*—Cooperative edge caching, multi-agent system, deep reinforcement learning, ultra-dense cellular networks.**

I. Introduction

DRIVEN by the increased penetration of smart devices, we have witnessed an unprecedented growth of mobile data traffic, which imposes a heavy traffic burden on the today's already congested cellular networks and backbones [1]. According to Cisco's report [2], the global mobile data traffic is expected to increase sevenfold over the period of 2017 to 2022. To accommodate these soaring traffic demands, the ultra-dense deployment of small cells in Next-generation Networks (NGN) allows the User Equipment (UE) to communicate using mmWave carriers, which substantially improves the wireless channel capacity, despite its potentially reduced power consumption [3]. However, the relatively slow backhaul links that connect the heterogeneous NGN base station (BS) [4], such as pico-cell base stations (PBSs), femto-cell access points (FAPs) and relays, to the network's backbone have become the bottleneck [5]. Qiu and Cao [6] have revealed that most of the backhaul bandwidth is occupied by multiple transmissions of some popular contents. Hence, mobile edge caching [7] is emerging as a promising technique of alleviating the traffic burden on the backhaul by offloading some popular contents to the edge of the network. Thereby, a large number of requests asking for the same contents can be directly accommodated by the edge caches, which shortens the content delivery distance and reduces the service delay [8].

However, the massive deployment of edge caches imposes additional challenges on the coordination of distributed cache placement. Due to the unique preference of each user, the content popularities in different BSs may present a somewhat surprising spatio-temporal difference [9]. This situation requires each BS to run a unique popularity evaluation algorithm, which is capable of learning and making its own caching decisions. However, the independent decision making of each BS would significantly reduce the efficiency of cache utilization [10], especially when the storage of a single BS is quite limited, since some popular contents may be redundantly cached by the BSs. In reality, the cached contents can be shared between different BSs through the Xn interface in 5G [4]. By avoiding the backbone and core network, the cost of fetching contents from the nearby BSs is much lower than transferring them from the content servers. When relying on content sharing, a BS may prefer not to cache a popular content which can be found in the nearby BSs, so that the limited edge storage can be utilized for caching some other contents and accommodate a larger number of content requests. Naturally, the caching decisions of different BSs are tightly intertwined with each other. However, each BS only knows its own caching decision, rather than the decisions of other BSs, since they rely on synchronous decision making. Moreover, we have to decide not only what but also where to cache in the geographically separate edge caches. The dimension of the caching action increases with the number of BSs and contents [11], which inevitably enhances the complexity of the problem to be solved. These obstacles make it harder to coordinate the massive number of BSs in ultra-dense NGN for efficiently

This work is supported by the Anhui Provincial Natural Science Foundation (No. 1908085QF266), the Fundamental Research Funds for the Central Universities (No. WK2100000009), the Youth Innovation Promotion Association CAS (No. CX2100107001) and the Major Science and Technology Project of Anhui (No. 201903a05020040). L. Hanzo would like to acknowledge the financial support of the Engineering and Physical Sciences Research Council projects EP/N004558/1, EP/P034284/1, EP/P034284/1, EP/P003990/1 (COALESCE), of the Royal Society's Global Challenges Research Fund Grant as well as of the European Research Council's Advanced Fellow Grant QuantCom.

Shuangwu Chen, Zhen Yao, Xiaofeng Jiang and Jian Yang are with the School of Information Science and Technology, University of Science and Technology of China, Hefei 230027, Anhui, China. E-mail: jianyang@ustc.edu.cn.

Lajos Hanzo is with the Department of Electronics and Computer Science, University of Southampton, SO17 1BJ, UK. E-mail:lh@ecs.soton.ac.uk.

TABLE I
COMPARISON OF THE EXEMPLARY RELATED WORKS ON EDGE CACHING.

References	Cooperative Caching (Content Sharing)	Distributed Decision	Variations of user distribution	Learning-based	Heterogeneous Content Size	Content Popularity Model	Solving Method
[14-15]	×	×	×	×	✓	stationary	convex programming
[16]	✓	×	×	×	×	stationary	integer-linear programming
[17]	✓	×	×	✓	×	spatial dynamics	transfer learning
[18-19]	×	×	×	×	✓	Spatial dynamics	Lyapunov optimization
[20]	✓	×	×	×	×	spatio-temporal dynamics	echo state network
[21]	✓	×	×	✓	×	spatio-temporal dynamics	reinforcement learning
[22-23]	×	×	×	✓	×	temporal dynamics	deep reinforcement learning
[24]	✓	×	×	×	✓	temporal dynamics	deep deterministic policy gradient
[25]	×	✓	×	✓	✓	spatio-temporal dynamics	deep reinforcement learning
[26]	×	✓	×	✓	×	spatio-temporal dynamics	multi-agent Q learning
[27]	✓	✓	×	✓	×	stationary	convex optimization
Proposed	✓	✓	✓	✓	✓	spatio-temporal dynamics	multi-agent deep reinforcement learning

exploiting the distributed edge caches [12].

To overcome these obstacles, most of the existing research in cooperative edge caching has been focused on optimizing the cache placement based on centralized decision making. In [13], a joint optimization problem was formulated by Gregori et al. for determining the optimal caching and transmission policies. By assuming that the user preferences are known in advance, the problem was addressed using finite-dimensional convex programming. By dynamically coordinating the user association, Hachem et al. [14] struck a trade off between the transmission cost and the storage cost. For a given popularity profile, Jiang et al. [15] derived an optimal cooperative content caching and transmission policy based on an integer-linear programming formulation. Although the above mentioned contributions have improved the cache hit rate or reduced the transmission delay, they relied on the idealistic assumption that the content popularity was known a *priori*. To avoid this assumption, Bharath et al. [16] conceived a transfer learning-based approach for each BS for estimating the unknown popularity profiles. However, their estimation technique relied on a training set, which might have different popularity-profiles from that of the actual content popularity. In [17], Kwak et al. formulated the content caching problem as a Lyapunov optimization problem by modeling the dynamics of the content requests as a virtual queue, which maximized the transmission rate, whilst satisfying a specific constraint on the max service delay. Kwak et al. [18] also made a step forward by jointly considering the content caching and BS association problem. In [19], the content popularity distribution was estimated by Chen et al. using an echo state network, while only relying on a limited number of request distribution samples and a sublinear algorithm was proposed for coordinating the content placement. In [20], the content requests were modeled as Markov processes by Sadeghi et al. and a reinforcement learning algorithm was proposed for finding the optimal caching strategy. Motivated by the recent advances in artificial intelligence, several recent studies solved the above problem by using the popular deep reinforcement learning (DRL) method. In [21] and [22], a deep Q-learning network (DQN) was used for making caching decisions. Since the DQN was unable to handle a large action space, a cooperative edge caching scheme was proposed by Qiao et al. [23] relying on a deep deterministic policy gradient (DDPG) model. In summary, by collecting all information from the distributed BSs, the centralized decision making failed to deal with the cooperative edge caching problem having an excessive search-space, given the high number of available contents and BSs.

Despite its importance, there is a paucity of literature on distributed edge caching. In [24], Jiang et al. proposed a multi-agent reinforcement learning framework for finding the optimal caching policy, when the state transition probabilities were unknown. Due to the large size of the state and action space, maintaining the Q-value for every action might exhaust the memory of each BS. Hence Zhong et al. [25] proposed a actor-critic DRL framework for decentralizing the caching decisions, where each actor (i.e., BS) made its own decision independently and a critic was used for estimating the overall reward. However, the edge caches would be under-utilized, since the contents sharing between the edge caches was not taken into account. The caching decision of each BS was purely based on the local observations without considering the actions and states of the cooperating BSs. In [26], a multi-armed bandit method was used by Song et al. for estimating the unknown content popularity and then a distributed caching algorithm was proposed based on the alternating direction method of multipliers. In order to coordinate the caching decisions, each BS had to broadcast its local state to all BSs, which would cause a high communication overhead among

BSs. The multi-armed bandit based popularity estimation assumed the popularity distribution to be stationary, which might not always be true in practice. Additionally, in order to find the optimal caching strategy, the algorithm in [26] had to iterate over all feasible actions which might lead to a high computation complexity. We compare the key characteristics of the related studies and our current work on edge caching in Table I.

However, the aforementioned solutions tend to suffer either from a high computational complexity or from a high exchange overhead between BSs, both of which actually increase with the number of BSs and the amount of contents. This problem is aggravated by ultra-dense cellular networks having a large number of BSs. Hence, we circumvent these impediments by designing a distributed cooperative caching algorithm for coordinating the caching decisions of the BSs, at a low information exchange overhead. Due to the movement of UEs, a user may not be able to completely download the whole content while crossing a small cell. We solve this problem by conceiving a coded caching system [27] in which each video clip is chopped into many small segments and can be flexibly requested by the UEs distributed across the coverage area of the BSs. Explicitly, we have to decide what, where and how many segments to cache in the edge caching system. Due to the fact that each BS is only directly aware of its local observation, the problem is formulated as a partially observable Markov decision process (POMDP) [28] based multi-agent decision problem, which jointly optimizes the costs of fetching contents from the local BS, from the nearby BSs and from the remote servers. In order to coordinate the caching decisions of a massive set of BSs, we devise a multi-agent actor-critic cooperation framework, where the variations rather than the full states of the environment are extracted and shared among the BSs. To deal with the large state and action space, we propose a cooperative edge caching algorithm where each BS makes its own caching decision based on both the local and global states. The contributions of this paper are summarized as follows:

- We formulate the cooperative edge caching problem as a POMDP based multi-agent decision problem, that maximizes the cumulative discounted reward for all edge caches. This formulation ensures the coordination of the edge caches by jointly optimizing the cost of content caching in the local BS, the cost of content sharing between the BSs and the cost of content retrieval from the servers.
- We devise a multi-agent actor-critic framework for our cooperative edge caching problem, where we spot and share any changes rather than handling the entire full dimensional environment, which significantly reduces the data exchange amongst the agents.
- By considering the time-dependent popularity distribution variations, we propose a variational recurrent neural network (VRNN) [29] based popularity estimation algorithm capable of learning without any prior knowledge of the user-preferences. To handle the time-variant geographic distribution of the UEs in the BSs' coverage area, we develop a long short term memory (LSTM) [30] based access state estimation model. Based on our powerful multi-agent deep reinforcement learning technique, we develop a cooperative edge caching algorithm, where the distributed decision making mitigates the complexity of problem solving.
- We evaluate the performance of our proposed algorithm using the real UE movement trajectories in a large-scale cellular network relying on numerous BSs and rich contents. The experimental results verify that our algorithm beneficially reduces the content retrieval and improves the edge caching hit rate by facilitating the collaboration of the edge caches.

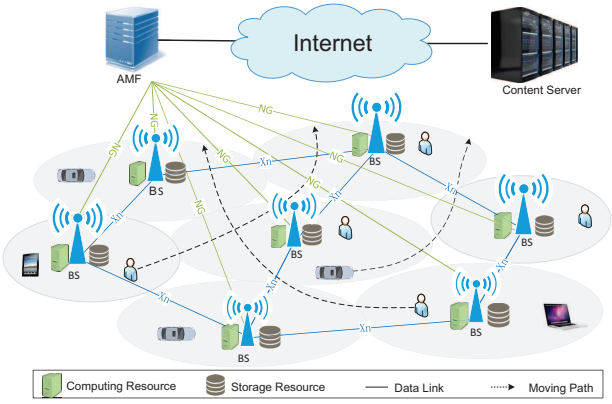


Fig. 1. Cooperative caching system over ultra-dense small-cell cellular networks.

The rest of this paper is organized as follows. Section II introduces our system model and problem formulation. In Section III, we propose a cooperative edge caching algorithm based on multi-agent deep reinforcement learning. The simulation results are discussed in Section IV. Finally, a conclusive discussion is offered in Section V.

II. System Model and Problem Formulation

Let us now discuss our system model, and then formulate the cooperative edge caching problem as a POMDP based multi-agent decision problem.

A. System model

We consider a typical content delivery system for the UEs of a NGN, which is illustrated in Fig. 1. The ultra-dense deployment of small cells allows the UEs to communicate at mmWave frequencies, which provides a high channel capacity but at a short coverage range. All BSs are connected to the NGN through backhaul links by means of the next generation (NG) interface [4]. Equipped with a memory device, each BS has a certain storage capacity so that some popular contents can be offloaded to the edge of the network. The UEs can fetch the contents directly from content servers through the core network. Alternatively, the same content may also be readily accommodated by the edge caches. The edge caching pushes the contents closer to the UEs, which significantly mitigates the traffic burden imposed on the backhaul and core

TABLE II
NOTATION USED IN THE PAPER

$b \in \mathcal{B}$	index of edge BS
$f \in \mathcal{F}$	index of content file
N	caching capacity of BS b
n_f	number of segments required for recovering the content f
$c_{b,f}$	number of segments of the content f in BS b
$\boldsymbol{c_b}$	cache state of BS b
m_b	number of UEs accessing BS b during time slot t
w_b	average sojourn time for the UEs in BS b
$\boldsymbol{v_b}$	access state of BS b
$p_{b,f}$	popularity of the content f in BS b
$\boldsymbol{x_b}$	request state of BS b
$a_{b,f}$	number of segment allocated to the content f in BS b
$\boldsymbol{a_b}$	action taken by BS b
$\boldsymbol{o_b}$	local state of BS b
R	immediate reward for all edge caches
$\boldsymbol{g}$	global state of the system
φ^{com}	function estimated by the communication module
φ^{dec}	function estimated by the decision module
φ^{cri}	function estimated by the critic module

networks, hence reducing the delivery delay. Although the storage space of a single BS is actually quite limited, the cached contents can be shared among the BSs through the Xn interface [4], which enables us to coordinate the BSs to take full advantage of the distributed edge caches. The user plane function (UPF) of NGNs would forward the user's request either to the central server or to the edge caches according to the contents placement. Therefore, the essential problem is to develop an efficient cooperative content caching and sharing strategy, so that the cost of fetching the requested contents can be minimized, which is also the optimization objective of our paper. The mathematical models of the aforementioned system can be described as follows.

1) Content caching model: Due to the short propagation distance of mmWave carriers, the UEs may only retrieve a portion of the requested contents from the BS accessed during a short sojourn time. Therefore, there is no need to retain the entire content file while conducting edge caching. Assume that each content is encoded by the rateless Fountain code [27] into multiple segments, so that a UE may recover the requested contents by collecting sufficient segments. Let $\mathcal{F} = \{1, \cdots, f, \cdots, F\}$ denote the set of contents files requested by the UEs. All the F contents can be retrieved from the remote content server. Different contents consist of different numbers of segments, each of which has the same segment size. Let n_f represent the minimum number of different segments required, so that the original content f can be successfully recovered. Actually, since different segments of the same content may be distributed across many BSs, there may be more than n_f segments in the edge caches for the content f, but for simplicity, we assume that there is no duplication of segments in the edge caches.

The set of BSs supporting edge caching is denoted by $\mathcal{B} = \{1, \cdots, b, \cdots, B\}$. For simplicity, we only consider the scenario, where a UE is associated with a single BS. This association is usually based on the distance between the UE and the BS. Thereby, the content sharing among the BSs has to rely on backhaul resources. We assume that each BS has the same caching capacity which can afford storing N segments. Indeed, the limited storage of a single BS is clearly not enough to accommodate all the contents, hence we have $N \ll \sum_{f=1}^{F} n_f$. The content placement is refreshed periodically and the time slot is indexed by $t = 0, 1, \cdots$. Since in a stationary scenario the content popularity changes slowly with time, the slot duration is long enough to complete the download of the updated contents from the content server. Let $c_{b,f}(t)$ denote the number of segments of the content f cached in the BS b. Naturally, there is no need to cache over n_f segments for a single content in each BS:

$$c_{b,f}(t) \leq n_f, \forall b \in \mathcal{B}, f \in \mathcal{F}. \tag{1}$$

Meanwhile, the total number of segments that have been cached in BS b has to satisfy the constraint

$$\sum\nolimits_{f \in \mathcal{F}} c_{b,f}(t) \leq N, \forall b \in \mathcal{B}. \tag{2}$$

The local cache state of BS b is defined as $\boldsymbol{c_b}(t) = \{\frac{c_{b,1}(t)}{N}, \cdots, \frac{c_{b,F}(t)}{N}\}$, where $\frac{c_{b,f}(t)}{N} \in [0, 1]$.

2) Content request model: The requested contents can be transmitted from either the edge caches or the content server. Let $\hat{p}_{b,f}(t)$ denote the number of requests for the content f received by BS b during the time slot t. The popularity of the content f can be approximated by its request probability, which is as follows

$$p_{b,f}(t) = \frac{\hat{p}_{b,f}(t)}{\sum_{f \in \mathcal{F}} \hat{p}_{b,f}(t)}. \tag{3}$$

Naturally, the content having a higher request probability is more popular among the users, and thus it is more likely to be cached in the edge. Actually, we have to decide not only which contents but also how many segments of a content file has to be cached. Hence, the number of segments requested by the users also has to be considered. We assume that the users in BS b request $\hat{q}_{b,f}(t)$ segments of the content f during slot t. For simplicity, we normalize $\hat{q}_{b,f}(t)$ to a range of $[0, 1]$:

$$q_{b,f}(t) = \frac{\hat{q}_{b,f}(t)}{q_{max}}, \tag{4}$$

where q_{max} is the maximum value that the number of requested segments may reach in a slot. The content request state of BS b is defined as $\boldsymbol{x_b}(t) = \{\boldsymbol{p_b}(t), \boldsymbol{q_b}(t)\}$, where $\boldsymbol{p_b}(t) = \{p_{b,1}(t), \cdots, p_{b,F}(t)\}$ and $\boldsymbol{q_b}(t) = \{q_{b,1}(t), \cdots, q_{b,F}(t)\}$.

3) User access model: Due to the stochastic nature of a user's movement, the user distribution may exhibit geographic differences between the various BSs. We assume that the UEs have the same chance of requesting a specific content. Naturally, having more access users or longer sojourn time may lead to more content requests. Due to the inhomogeneous distribution of the UEs in the BSs' coverage area, the content requests may exhibit regional deviations. In order to characterize the effect of the non-uniform user distribution on content requests, we have to model the user access process. The UEs subscribing to the content delivery service compose a set $\mathcal{E} = \{1, \cdots, e, \cdots, E\}$. In a time-slotted system, a mobile UE may cross the covered area of multiple BSs during a time slot. Let $w_{b,e}(t)$ denote the sojourn time of the UE $e \in \mathcal{E}$ in the coverage area of BS b during the time slot t, which is maintained by the access and mobility management function

(AMF) in NGNs [31]. The total number of UEs that have access to BS b can be represented by

$$m_b(t) = \sum\nolimits_{e\in\mathcal{E}} I[w_{b,e}(t)], \tag{5}$$

where $I(\cdot)$ is an indicator function which is given by

$$I(x) = \begin{cases} 1, & \text{if } x > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

Specifically, $I(w_{b,e}(t)) = 0$ means that the UE e has no access to BS b within the duration of $[t, t+1)$. Here, we can also define the UEs' average sojourn time in BS b as

$$w_b(t) = \frac{\sum_{e\in\mathcal{E}} w_{b,e}(t)}{m_b(t)}. \tag{7}$$

Here, we normalize $\bar{m}_b$ and $\bar{m}_b$ to the range of $[0, 1]$:

$$\bar{m}_b = \frac{m_b}{M}, \qquad \bar{w}_b = \frac{w_b}{\Delta t}, \tag{8}$$

where M is the maximum number of cache users, and Δt is the time duration of a slot which is also the maximum sojourn time in a slot. Therefore, the access state of any BS b in the t-th time slot can be represented by $\boldsymbol{v_b}(t) = \{\bar{m}_b(t), \bar{m}_b(t)\}$.

B. Problem Formulation

Since the states of the cache placement, the content request and the user access at the next moment are only related to the current states and the caching decision, but they have nothing to do with the earlier states, we can model the evolution of the states by a Markov process. Each BS can explicitly know its local cache placement $\boldsymbol{c_b}(t)$ while making a caching decision at the start of time slot t. However, because the request state $\boldsymbol{x_b}(t)$ and the access state $\boldsymbol{v_b}(t)$ are two statistics, which respectively reflect the content popularity and the user distribution during the time interval $[t, t+1)$, neither of them can be observed until the end of time slot t. The fact that the full information concerning the states is not completely observable at the time of making caching decisions motivates us to formulate our caching decision problem as a POMDP. We define the states, actions, observations and rewards as follows.

1) State: The local state $\boldsymbol{s_b}(t)$ of BS b at time slot t is defined as

$$\boldsymbol{s_b}(t) = \{\boldsymbol{c_b}(t), \boldsymbol{x_b}(t), \boldsymbol{v_b}(t)\} \tag{9}$$

which consists of three parts, including the local cache state $\boldsymbol{c_b}$, access state $\boldsymbol{v_b}$ and request state $\boldsymbol{x_b}$. Then, the state of the whole system is defined as $\boldsymbol{s}(t) = \{\boldsymbol{s_1}(t), \cdots, \boldsymbol{s_b}(t), \cdots, \boldsymbol{s_B}(t)\}$.

2) Action: In order to accommodate the dynamic changes of the content popularity, each BS is expected to actively adjust its local cache placement by removing some unpopular contents or adding other popular contents. Let $a_{b,f}$ denote the cache size (quantified by the number of segments) that BS b decides to assign to the content f while observing the state $\boldsymbol{o}(t)$. At the time instant t, let us assume that there are $c_{b,f}(t)$ segments of the content f in the BS b. While taking the action $a_{b,f}(t)$, BS b assigns $a_{b,f}(t)$ segments for the content f. Compared to the slot duration, the time required for updating the cache replacement is usually trivial. Hence, at time instant $t+1$, there are $a_{b,f}(t)$ segments for the content f in the BS b, i.e. we have $\boldsymbol{c_b}(t+1) = \boldsymbol{a_b}(t)$. In order to indicate the changes of the cache placement after taking the action $a_{b,f}$, we compare $a_{b,f}$ to $c_{b,f}$. Specifically, if we have $a_{b,f} > c_{b,f}$, then $|a_{b,f} - c_{b,f}|$ segments of the content f are added to the BS b. If we have $a_{b,f} > c_{b,f}$, then $|a_{b,f} - c_{b,f}|$ segments of the content f are removed from BS b. Otherwise, there is no adjustment of the content f. At time instant t, the action taken by BS b is defined as

$$\boldsymbol{a_b}(t) = \{a_{b,1}(t), \cdots, a_{b,f}(t), \cdots, a_{b,F}(t)\}. \tag{10}$$

Notably, the action $a_{b,f}$ should also meet the constraints (1) and (2). The action for the whole system is defined as $\boldsymbol{a} = \{\boldsymbol{a_1}, \cdots, \boldsymbol{a_b}, \cdots, \boldsymbol{a_B}\}$.

3) Observation: At the decision time t, the BSs cannot be explicitly aware of the current content request state $\boldsymbol{x}_b(t)$ and user access state $\boldsymbol{v}_b(t)$, but can readily acquire their previous states $\boldsymbol{x}_b(t-1)$ and $\boldsymbol{v}_b(t-1)$ by counting the number of requests for each content, the number of access UEs and the average sojourn time during the time interval $[t-1, t)$. Due to the time-dependent nature of the state dynamics, we can estimate the current states of user access and content request based on the historical state sequences $\{\boldsymbol{x_b}(t-1), \boldsymbol{x_b}(t-2), \cdots\}$ and $\{\boldsymbol{v_b}(t-1), \boldsymbol{v_b}(t-2), \cdots\}$. Hence, the observation is defined as the state estimation at time instant t

$$\boldsymbol{o_b}(t) = \{\boldsymbol{c_b}(t), \widetilde{\boldsymbol{x}}_{\boldsymbol{b}}(t), \widetilde{\boldsymbol{v}}_{\boldsymbol{b}}(t)\}, \tag{11}$$

where $\widetilde{\boldsymbol{x}}_{\boldsymbol{b}}(t)$ and $\widetilde{\boldsymbol{v}}_{\boldsymbol{b}}(t)$ denote the estimates of the request state and the access state, specifically. The observation for the whole system is denoted by $\boldsymbol{o} = \{\boldsymbol{o_1}, \cdots, \boldsymbol{o_b}, \cdots, \boldsymbol{o_B}\}$.

4) Reward: In the cooperative caching system, the user requests may be accommodated by the content server, local BS or nearby BS as depicted in Fig. 2, depending on the content placements. In particular, the BS currently being accessed by the UE is called the local BS, while the other BSs are termed as nearby BS. The three optional ways of fetching the content often correspond to different costs, which are listed as follows. In practice, the cost may be defined in different forms such as monetary cost, transmission delay, bandwidth requirement or a combination of them.

(a) If the requested contents have been cached in the local BS, they can be delivered to the UEs directly using a high bandwidth and low latency. Let α_b denote the cost of delivering a segment from the local BS b. Assume that BS b has fetched $r_b^b(t)$ segments from its local cache during the time slot t. Thereby, the cost of service by the local BS is given by $\alpha_b r_b^b(t)$.

(b) In the cooperative caching system, the segments belonging to the same content may be distributed to multiple BSs. If the local BS has insufficient segments for the requested content but nearby BSs keep them, the user requests can be accommodated by sharing the contents among those BSs. Due to the fairly limited cache space, if a requested content segment is fetched from a nearby BS, it will be directly forwarded to the user. Let β_b^k $(k \in \mathcal{B}, k \neq b)$ denote the cost of delivering a segment from the nearby BS k to a UE, which accesses the BS b. Since the contents sharing

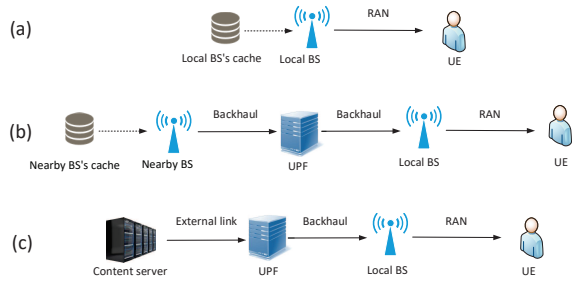


Fig. 2. The transmission paths for different content placement.

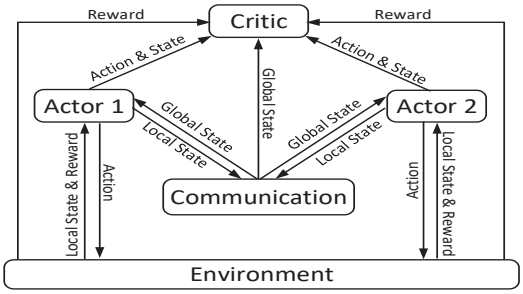


Fig. 3. The multi-agent actor-critic cooperation framework.

among the distributed BSs would consume the backhaul resource, the cost of fetching segments from nearby BSs is much higher than that from the local BS, *i.e.* we have $\beta_b^k > \alpha_b$. Notably, the value of β_b^k depends on the distance between BS b and k. Assume that BS b has fetched $r_b^k(t)$ segments from BS k in time slot t. Thereby, the cost of service by the nearby BSs $\sum_{k\in\mathcal{B},k\neq b}\beta_b^k r_b^k(t)$.

(c) If the requested content is not offloaded to the edge caches, the user would fetch it from the content server. Let θ_b denote the cost of delivering a segment from the original server to a UE, which accesses the BS b. Due to the resource consumption of both the backhaul and the core network, the cost of serving by the content server is much higher than that by the edge caches, *i.e.* we have $\theta_b > \beta_b^k$. Here, we refer to 0 as the index of the content server. Assume that r_b^0 segments are transmitted from the content server during time slot t. Then the cost of service by the content server is $\theta_b r_b^0(t)$.

Given the above, the cost of content delivery during time slot t can be represented by

$$\alpha_b r_b^b(t) + \sum\nolimits_{k\in\mathcal{B},k\neq b}\beta_b^k r_b^k(t) + \theta_b r_b^0(t). \tag{12}$$

Additionally, after making a decision to adjust its local cache, the BS removes some of the less popular contents and adds more popular contents. We assume that all the added contents are downloaded from the content server. The cache replacement may impose additional backhaul requirements on the core network. Therefore, the total cost of content delivery relying on edge caches should contain not only the cost of transferring data to UEs from the local BS, nearby BSs and content server, but also the cost of updating the cache placement. During time slot t, the number of segments replaced in BS b can be represented by $\sum_{f\in\mathcal{F}}[a_{b,f}(t) - c_{b,f}(t)]^+$, where we define $(x)^+ \triangleq \max(x, 0)$. Let δ_b denote the cost of replacing a segment in BS b. The cost of cache replacement is

$$\sum\nolimits_{f\in\mathcal{F}}\delta_b[a_{b,f}(t) - c_{b,f}(t)]^+. \tag{13}$$

Upon using edge caches, the total cost is a sum of (12) and (13).

Without edge caches, all user requests have to be handled by the content servers and the corresponding cost is $\sum_{k\in\mathcal{B}}\theta_b r_b^k(t) + \theta_b r_b^0(t)$. By pushing the contents closer to the users, the edge cache can help reduce the cost of content delivery. Hence, the cost associated with using edge caches is usually much lower than without. The higher the cost saving, the more effective the edge caching. We can define the immediate reward of BS b as the cost saving attained by edge caching, which is formulated as:

$$\begin{aligned} R_b(t) =&(\theta_b - \alpha_b)r_b^b(t) + \sum\nolimits_{k\in\mathcal{B},k\neq b}(\theta_b - \beta_b^k)r_b^k(t) \\ &- \sum\nolimits_{f\in\mathcal{F}}\delta_b(a_{b,f}(t) - c_{b,f}(t))^+, \end{aligned} \tag{14}$$

where the parameters $\alpha_b, \beta_b, \theta_b$ and δ_b are predefined constants, and r_b^k ($k \in \mathcal{B}$) is readily obtained by evaluating the statistics of the number of segments transmitted from BS k during the time interval $[t, t+1)$. We cannot calculate the reward until the end of time slot t. Maximizing this reward also corresponds to minimizing the cost of content delivery with edge caches. Due to the existence of r_b^k, the immediate reward of BS b depends both on its own cache placement as well as on the cache placement of nearby BSs. Then, the immediate reward for the whole system is given by

$$R(t) = \sum\nolimits_{b\in\mathcal{B}} R_b(t). \tag{15}$$

The model formulated can also be applied to the multiple-input-multiple-output (MIMO) scenario [32], [33], where a single UE can simultaneously establish multiple connections with its nearby BSs. With the aid of MIMO schemes, the UE can directly fetch contents both from the local and from the nearby BSs, without requiring any backhaul resources. Hence, the cost of service by nearby BSs may be the same as that by the local BS.

In the distributed edge caching system, each BS can be considered as an agent, which has to decide its own cache placement according to the system states. Let $\pi = \{\pi_1, \cdots, \pi_B\}$ represent the caching strategy, which maps a state $\boldsymbol{s}$ to a legitimate action $\boldsymbol{a}$, i.e $\boldsymbol{a} = \pi(\boldsymbol{s})$. Since the agent's action has an impact both on the immediate reward as well as on the long-term reward, all agents are expected to work cooperatively to find an optimal strategy π^* that maximizes the long-term reward. Here, we assume that the long-term reward is discounted by a discount factor of $\gamma \in (0, 1)$. Then, the cooperative caching problem can be formulated as a multi-agent decision problem that maximizes the cumulative discounted reward:

$$\begin{aligned} \max_{\pi} \quad & V^{\pi}(\boldsymbol{s}) = \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t R(t) \mid \boldsymbol{s}(0) = \boldsymbol{s}, \pi\right], \\ \text{s.t.} \quad & (1) \text{ and } (2), \end{aligned} \tag{16}$$

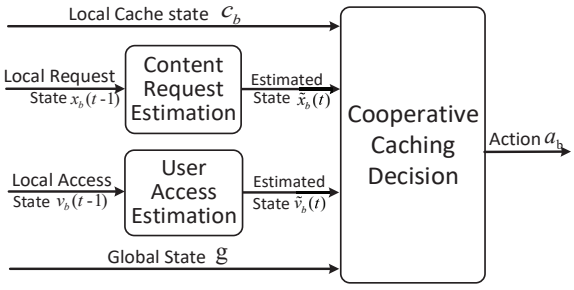


Fig. 4. The model architecture of each actor.

when starting from an initial state $\boldsymbol{s}$, where $V^{\pi}(\boldsymbol{s})$ is the state value function. Given its Markovian nature, the optimal strategy π^* follows Bellman's function:

$$V^{\pi^*}(\boldsymbol{s}) = R(\boldsymbol{s}, \pi^*(\boldsymbol{s})) + \gamma \sum\nolimits_{\boldsymbol{s}' \in \mathcal{S}} P_{\boldsymbol{s}\boldsymbol{s}'} V^{\pi^*}(\boldsymbol{s}'), \qquad (17)$$

where $P_{\boldsymbol{s}\boldsymbol{s}'}$ is the state transition probability from $\boldsymbol{s}$ to $\boldsymbol{s}'$. However, without any prior knowledge, $P_{\boldsymbol{s}\boldsymbol{s}'}$ is actually unknown.

Due to the unique preference of each user, the requests received by different BSs may present a remarkable difference. Each BS is expected to run a unique popularity learning algorithm and makes its own decisions as to which contents and how many segments should be stored in its local cache. Meanwhile, because the cached contents can be shared among the BSs, the caching decisions of different BSs are tightly intertwined with each other. Particularly, the BS may not cache any segments of a popular content, which can be found in the nearby BSs, so that the limited edge storage can be utilized to support a larger number of user request by caching some other contents. Hence, the independent decision making of a specific BS is also affected by the states and actions of other BSs. However, each BS only explicitly know its local observation rather than the full states and actions of the environment. To address this problem, in the next section, we will introduce a multi-agent deep learning based caching algorithm.

III. Multi-agent Deep Reinforcement Learning based Cooperative Edge Caching

Motivated by the recent advances in artificial intelligence, we present a multi-agent actor-critic framework for cooperative edge caching. Although the local state is not completely observable, we can observe the states sequence of previous intervals. Aided by the time-dependent nature of the state dynamics, we can estimate the current states of user access and content request based on the historical observations. Hence, we develop an LSTM based access state estimation model and a VRNN based request state estimation model. In order compress and share the global states among the BSs, a communication model is introduced. Finally, we propose a cooperative edge caching algorithm based on multi-agent deep reinforcement learning.

A. Multi-Agent Actor-Critic Cooperation Framework

The multi-agent actor-critic cooperation framework for edge caching is presented in Fig. 3, which consists of three modules: actor, critic and communication.

1) Actor: Each agent has an actor network, which learns a unique strategy function π that maps the state, including the local observation $\boldsymbol{s_b}$ and the global state $\boldsymbol{g}$ shared by the communication module, to a legitimate action. However, as depicted in the formulations, the local access state $\boldsymbol{v_b}(t)$ and the local request state $\boldsymbol{x_b}(t)$ are not explicitly known at the time of making the caching decisions. Fortunately, the states of previous slots can be readily acquired. Due to the temporal dependence of the states variation, this inspires us to estimate the current state based on the historical state sequences. As shown in Fig. 4, we develop a user access estimation model as well as a content request estimation model for estimating the current request state $\widetilde{\boldsymbol{x}}_{\boldsymbol{b}}(t)$ and the current access state $\widetilde{\boldsymbol{v}}_{\boldsymbol{b}}(t)$ based on the time sequences $\{\boldsymbol{x_b}(t-1), \boldsymbol{x_b}(t-2), \cdots\}$ and $\{\boldsymbol{v_b}(t-1), \boldsymbol{v_b}(t-2), \cdots\}$. Based on the estimated states, the caching decision module will decide which specific contents and how many segments are cached in the local BS.

2) Critic: The centralized critic network is used for estimating the action-value function, which gives the overall reward, while taking the action $\boldsymbol{a}$ based upon the observations $\boldsymbol{o}$ and the global state $\boldsymbol{g}$. In practice, we can deploy the critic network in the central NGN core network so that the observations and actions of all agents can be easily collected. The error between the critic output and the actual reward is used for updating the parameters of the actor networks.

3) Communication: In order to make the agents act cooperatively, the independent decision making of a specific agent is expected to take not only its local observations, but also the global state, i.e. the observations $\boldsymbol{o}$ and actions $\boldsymbol{a}$ of other agents, into account. A communication module is designed to collect and share the global state among all agents. However, since the number of full states of $\boldsymbol{a}$ and $\boldsymbol{o}$ expands with the number of agents and content files, the overhead of sharing the full states of $\boldsymbol{a}$ and $\boldsymbol{o}$ directly with each agent is actually unfeasible. Hence, the communication module has to reduce the dimensionality by encoding the observations and actions of the agents. Actually, the communication module can be also deployed in the NGN core network.

B. Request State Estimation

Since the content requests are usually continuous in time, the transitions of request state obey certain time-dependence. The temporal dependencies of the requested states make it possible to predict the variability of content request based on previous state sequence.

In our system, although the current state of content request $\boldsymbol{x}_b(t)$ is not completely observable at the decision time t, the states of the previous slots, i.e. $\{\boldsymbol{x_b}(t-1), \boldsymbol{x_b}(t-2), \cdots\}$, are exactly known. However, the dimension of the content request state, by definition, is proportional to the number of content files, which would be very large in practice. This makes it intractable to explicitly model the complex variability of the high-dimensional sequential data using standard RNNs, since the conditional output probability models of standard RNNs are usually based on a simple unimodal distribution [29]. Fortunately, because the Variational Autoencoder (VAE)

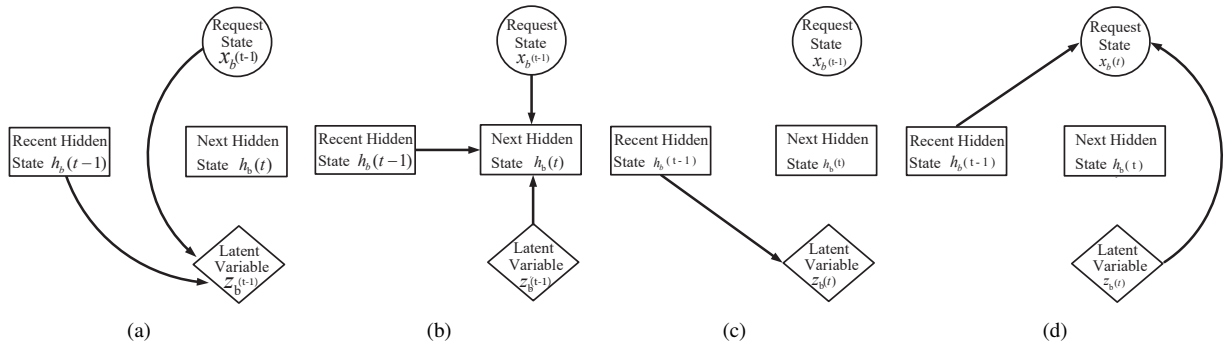


Fig. 5. The operations of VRNN-based content request prediction. (a) Encoding the observed states. (b) Updating the hidden states. (c) Generating the next latent variable. (d) Predicting the content request.

[34] is capable of modeling the data, which includes multi-modal conditional distributions, by extending the VAE into a recurrent framework, the VRNN offers an effective paradigm for characterizing the complex relationship among the output variables at different time steps. Hence, we develop a VRNN model for estimating the request state.

In time slot t, the VAE can encode the variations of the high-dimensional state $\boldsymbol{x_b}(t)$ into a set of latent random variables $\boldsymbol{z}_b$, which extracts the structural features of $\boldsymbol{x}_b$. The temporal dependencies between the latent random variables $\boldsymbol{z}_b$ across neighboring time slots are integrated into the RNN's hidden state $\boldsymbol{h}_b$. The operations of VRNN-based content request estimation are illustrated in Fig. 5, which are as follows.

1) Encoding the observed states: At time instant t, given the previous state $\boldsymbol{x}_b(t-1)$ and the recent hidden state $\boldsymbol{h}_b(t-1)$, the posterior on the latent variable is approximated by

$$\boldsymbol{z}_b(t-1)|\boldsymbol{x}_b(t-1) \sim \mathcal{N}[\boldsymbol{\mu}_{b,z}, \text{diag}(\boldsymbol{\sigma}^2_{b,z})], \tag{18}$$

where $(\boldsymbol{\mu}_{b,z}, \boldsymbol{\sigma}_{b,z}) = \varphi^{\text{enc}}[\varphi^x(\boldsymbol{x}_b(t-1)), \boldsymbol{h}_b(t-1)]$, while $\boldsymbol{\mu}_{b,z}$ and $\boldsymbol{\sigma}_{b,z}$ respectively denote the mean and variance, which can be estimated by a neural network $\varphi^{\text{enc}}(\cdot)$. Furthermore, $\varphi^x(\cdot)$ can also be a neural network, which extracts the features from $\boldsymbol{x}_b$. Since $\boldsymbol{h}_b(t-1)$ represents the historical variations of the sequence $\{\boldsymbol{x}_b(t-2), \boldsymbol{x}_b(t-3), \cdots, \}$, the encoding of $\boldsymbol{h}_b(t-1)$ and $\boldsymbol{x}_b(t-1)$ can extract the temporal dependencies into the latent random variables $\boldsymbol{z}_b$.

2) Updating the hidden states: The hidden state is updated using the recurrence equation:

$$\boldsymbol{h}_b(t) = f[\varphi^x(\boldsymbol{x}_b(t-1)), \varphi^z(\boldsymbol{z}_b(t)), \boldsymbol{h}_b(t-1)], \tag{19}$$

where $f(\cdot)$ is the transition function which can be estimated by an RNN, and $\varphi^z(\cdot)$ is also a neural network that extracts the features from $\boldsymbol{z}_b$.

3) Generating the next latent variable: The prior on the latent random variable follows the distribution:

$$z_b(t) \sim \mathcal{N}[\boldsymbol{\mu}_{b,0}, \text{diag}(\boldsymbol{\sigma}^2_{b,0})], \tag{20}$$

where $(\boldsymbol{\mu}_{b,0}, \boldsymbol{\sigma}_{b,0}) = \varphi^{\text{prior}}[\boldsymbol{h}_b(t)]$, while $\boldsymbol{\mu}_{b,0}$ and $\boldsymbol{\sigma}_{b,0}$ denote the mean and variance, which can be estimated by a neural network $\varphi^{\text{enc}}(\cdot)$. Given a $\boldsymbol{h}_b(t)$, we can generate a $\tilde{\boldsymbol{z}}_b(t)$ by randomly sampling the distribution in (20).

4) Predicting the content request: The conditional distribution of the content request state can be estimated by:

$$\tilde{\boldsymbol{x}}_b(t)|\boldsymbol{z}_b(t) = \mathcal{N}[\boldsymbol{\mu}_{b,x}, \text{diag}(\boldsymbol{\sigma}^2_{b,x})], \tag{21}$$

where $(\boldsymbol{\mu}_{b,x}, \boldsymbol{\sigma}_{b,x}) = \varphi^{\text{dec}}[\varphi^z(\boldsymbol{z}_b(t)), \boldsymbol{h}_b(t)]$, while $\boldsymbol{\mu}_{b,x}$ and $\boldsymbol{\sigma}_{b,x}$ are the parameters of the conditional distribution, which can be estimated by the decoder of the VAE. Given the generated latent variable $\tilde{\boldsymbol{z}}_b(t)$, we can infer the next content request state $\tilde{\boldsymbol{x}}_b(t)$.

Each BS relies on a VRNN model, which is trained by the unique traces of the historical content request received by the BS. The training objective function is to maximize the variational lower bound:

$$\mathbb{E}_{q(\boldsymbol{z}_b|\boldsymbol{x}_b)} \sum_{t=1}^{T} \Big[\log p(\boldsymbol{x}_b|\boldsymbol{z}_b) - KL(q(\boldsymbol{z}_b|\boldsymbol{x}_b)||p(\boldsymbol{z}_b) \Big], \tag{22}$$

where $p(\boldsymbol{x}_b|\boldsymbol{z}_b)$ is the conditional probability defined in (21) and $KL(\cdot)$ is the Kullback-Leibler divergence between the approximate posterior probability $q(\boldsymbol{z}_b|\boldsymbol{x}_b)$ defined in (18) and the prior probability $p(\boldsymbol{z}_b)$ defined in (20).

C. Access State Estimation

Since users are randomly distributed across the coverage area of the BSs, the access states exhibit geographic variations. Each BS employs a learning model to get its unique variation of the access states.

According to the research results in [35],the users' positions and movements are usually predictable based on their historical trajectory, which makes it possible to estimate the current access state $\boldsymbol{v}_b(t)$ based on the previous state sequence $\{\boldsymbol{v_b}(t-1), \boldsymbol{v_b}(t-2), \cdots\}$. Given that the access state $\boldsymbol{v}_b$ is only two-dimensional, we can adopt a simple LSTM networks to model the dynamics of the user access state, and to capture the temporal dependencies present in the state transitions.

Like conventional RNNs, a typical LSTM network has a chain structure which consists of an input layer, multiple hidden layers and an output layer. The difference is that the hidden layer is a memory block which consists of an input gate, a forget gate, multiple memory cells and an output gate. In LSTM, the output values are decided by both the current inputs and the previous inputs that are represented by the hidden states of the neural network.

At time instant t, the previous access state $\boldsymbol{v}_b(t-1)$ is used as the input of the LSTM network and the output is an estimate of the current access state. The parameters of the memory block are updated as follows:

1) *The Forget Gate* will selectively forget the input state. The activation vector is

$$\boldsymbol{f}_t = \sigma[\boldsymbol{W}_f \boldsymbol{v}_b(t-1) + \boldsymbol{U}_f \boldsymbol{h'}(t-1) + \boldsymbol{b}_f], \quad (23)$$

where $\boldsymbol{W}_f$ is the weight between the input and the forget gate, $\boldsymbol{U}_f$ is the weight between the previous hidden state $\boldsymbol{h'}(t-1)$ and the forget gate, $\boldsymbol{b}_f$ is the bias and $\sigma(\cdot)$ is the logistic sigmoid function.

2) *The Input Gate* will selectively remember the input states and it is updated by

$$\begin{aligned} \boldsymbol{i}_t &= \sigma[\boldsymbol{W}_i \boldsymbol{v}_b(t-1) + \boldsymbol{U}_i \boldsymbol{h'}(t-1) + \boldsymbol{b}_i], \\ \widetilde{\boldsymbol{C}}_t &= \tanh[\boldsymbol{W}_c \boldsymbol{v}_b(t-1) + \boldsymbol{U}_c \boldsymbol{h'}(t-1) + \boldsymbol{b}_c], \end{aligned} \quad (24)$$

where $\boldsymbol{W}_i$ and $\boldsymbol{W}_c$ are the weights between the input state $\boldsymbol{v}_b(t-1)$ and input gate, while $\boldsymbol{U}_i$ and $\boldsymbol{U}_c$ are the weights between the previous hidden state $\boldsymbol{h'}(t-1)$ and the input gate. Finally, $\tanh(\cdot)$ is the hyperbolic tangent function. The cell state vector is updated by

$$\boldsymbol{C}_t = \boldsymbol{f}_t \odot \boldsymbol{C}(t-1) + \boldsymbol{i}_t \odot \widetilde{\boldsymbol{C}}_t, \quad (25)$$

where $\odot$ denotes the Hadamard product operator.

3) *The Output Gate* will predict the next state. The hidden state is updated by

$$\begin{aligned} \boldsymbol{o}_t &= \sigma[\boldsymbol{W}_o \boldsymbol{v}_b(t) + \boldsymbol{U}_o \boldsymbol{h'}(t-1) + \boldsymbol{b}_o], \\ \boldsymbol{h'}(t) &= \boldsymbol{o}_t \odot tanh(\boldsymbol{C}_t), \end{aligned} \quad (26)$$

where $\boldsymbol{W}_o$ is the weight between the current input $\boldsymbol{v}_b(t)$ and the output gate, $\boldsymbol{U}_f$ is the weight between the previous hidden state $\boldsymbol{h'}(t-1)$ and the output gate. The next state is predicted by

$$\tilde{\boldsymbol{v}}_b(t) = \sigma[\boldsymbol{W}_t \boldsymbol{h'}(t)], \quad (27)$$

where $\boldsymbol{W}_t$ is the weight vector of the output gate.

The LSTM model is trained in advance using the historical access state sequences collected by the BSs. The error between the real state and the predicted state can be used for updating the weight parameters W and U.

D. Cooperative Caching Decision Based on Multi-Agent Deep Reinforcement Learning

Due to the synchronous decision makings of all agents, at time instant t, the agent b must independently take an action before knowing the actions and observations of other agents. Hence, the shared global state $\boldsymbol{g}$ is expected to represent the evolution of the actions and observations of all agents. For this purpose, the communication module adopts a LSTM network to extract the time-dependent variability of the actions and observations of all agents into its hidden state. We can define the global state as the hidden state of the LSTM network:

$$\boldsymbol{g}(t) = \varphi^{com}[\boldsymbol{o}(t), \boldsymbol{a}(t), \boldsymbol{g}(t-1); \phi^{com}], \quad (28)$$

where φ^{com} denotes the function estimated by the LSTM network and ϕ^{com} denotes the network parameters. The LSTM network will update the global state over time by collecting the recent observation and actions.

Additionally, sharing the global state $\boldsymbol{g}$ rather than the full states of $\boldsymbol{o}$ and $\boldsymbol{a}$ among the agents will significantly reduce the overhead of data interaction. Let G denote the dimension of $\boldsymbol{g}$. We assume that the data for each dimension can be represented as a float. The overhead of transferring the global state $\boldsymbol{g}$ to all the B agents is $B \cdot G$ floats. By contrast, without the communication module, exchanging the full states of $\boldsymbol{a}(t)$ and $\boldsymbol{o}(t)$ directly with all the B agents requires $B^2 \cdot (4F+2)$ floats, since the dimensions of $\boldsymbol{o}(t)$ and $\boldsymbol{a}(t)$ are $B \cdot (F + 2F + 2)$ and $B \cdot F$, respectively. Because the LSTM extracts the variation of the sequences $\boldsymbol{o}(t)$ and $\boldsymbol{a}(t)$ into $\boldsymbol{g}(t)$, the dimension of $\boldsymbol{g}(t)$ is usually much smaller than that of $\boldsymbol{o}(t)$ and $\boldsymbol{a}(t)$, i.e. we have $G \ll B \cdot (4F+2)$. Hence sharing the global state $\boldsymbol{g}$ rather than the full states of $\boldsymbol{o}$ and $\hat{\boldsymbol{a}}$ among the agents will significantly reduce the data interaction.

As shown in Fig. 4, given the global state $\boldsymbol{g}$ and the local observation $\boldsymbol{o}$, the remaining problem is to find the optimal caching action that maximizes the overall reward. To solve this problem, the caching decision module adopts a fully connected neural network to estimate the function relationship between the state and the action, which is as follows:

$$\hat{\boldsymbol{a}}_{\boldsymbol{b}}(t) = \varphi_b^{dec}[\boldsymbol{c}_{\boldsymbol{b}}(t), \tilde{\boldsymbol{x}}_{\boldsymbol{b}}(t), \tilde{\boldsymbol{v}}_{\boldsymbol{b}}(t), \boldsymbol{g}(t-1); \phi_b^{dec}], \quad (29)$$

where $\hat{\boldsymbol{a}}_{\boldsymbol{b}} = \{\hat{a}_{b,1}, \cdots, \hat{a}_{b,F}\}$ is the outputted action vector and ϕ_b^{dec} denotes the parameter of the neural network. Generally, we use a softmax layer in the output layer to normalize the output values to the range of $[0,1]$, that is $\hat{a}_{b,f} \in [0,1]$. Since the output vector $\hat{\boldsymbol{a}}_{\boldsymbol{b}}$ of the actor network is continuous, the actual action performed by each BS has to be discretized during the execution phase, which is given by

$$\boldsymbol{a}_{\boldsymbol{b}} = \lfloor \hat{\boldsymbol{a}}_{\boldsymbol{b}} N \rfloor = \{\lfloor \hat{a}_{b,1} N \rfloor, \cdots, \lfloor \hat{a}_{b,F} N \rfloor\} \quad (30)$$

where $\lfloor \cdot \rfloor$ is the rounding down operation. Naturally, $a_{b,f} = 0$ means that BS b decides not to retain the content f in its local cache.

Taking the three modules in Fig. 4 as a whole, the actor adopts an integrated neural network to estimate the strategy function that maps the current local observation $\boldsymbol{o}_{\boldsymbol{b}}(t)$ and the previous global state $\boldsymbol{g}(t-1)$ to the action vector $\hat{\boldsymbol{a}}_{\boldsymbol{b}}$, which can be represented by

$$\begin{aligned} \hat{\boldsymbol{a}}_{\boldsymbol{b}}(t) &= \varphi_b[\boldsymbol{c}_{\boldsymbol{b}}(t), \boldsymbol{x}_{\boldsymbol{b}}(t-1), \boldsymbol{v}_{\boldsymbol{b}}(t-1), \boldsymbol{g}(t-1); \phi_b] \\ &= \varphi_b[\boldsymbol{o}_{\boldsymbol{b}}(t), \boldsymbol{g}(t-1); \phi_b], \end{aligned} \quad (31)$$

where ϕ_b denotes the parameter of the actor network.

In order to assess the effects of the actions, the centralized critic module adopts a fully connected neural network to estimate the action-value function, which is as follows:

$$Q[\boldsymbol{o}(t), \boldsymbol{g}(t-1), \boldsymbol{a}(t)] = \varphi^{cri}[\boldsymbol{o}(t), \boldsymbol{g}(t-1), \hat{\boldsymbol{a}}(t)], \quad (32)$$

where φ^{cri} is a neural network that gives the long-term reward V after taking the actions $\hat{\boldsymbol{a}}$ upon the observations $\boldsymbol{o}$.

In practice, although the actual system parameters may be different from what we expect, the parameters of the multi-agent actor-critic model can be updated as depicted in Fig. 6. In order to speed up the training process, we usually train the

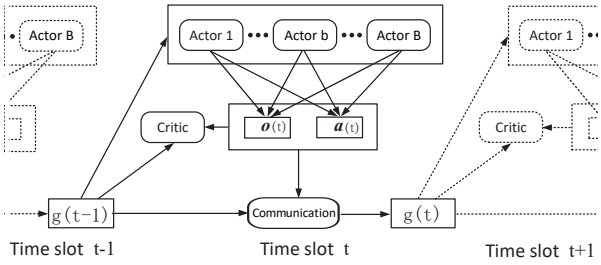


Fig. 6. The training process of the multi-agent actor-critic model.

model using the historical data set before deploying it online. In time slot t, the agents send the local observations $\boldsymbol{o}(t)$ along with the actions $\hat{\boldsymbol{a}}(t)$ taken by each actor to the critic. After executing the actions in the environment, the immediate reward $R(t)$ and the subsequent observations $\boldsymbol{o}(t+1)$ are fed back to the critic. Then, the parameters of the critic network are updated by minimizing the least squares temporal difference:

$$L(\varphi^{cri}) = \mathbb{E}\left\{y(t) - \varphi^{cri}[\boldsymbol{o}(t), \boldsymbol{g}(t-1), \hat{\boldsymbol{a}}(t)]\right\}^2, \quad (33)$$

where we have $y(t) = R(t) + \gamma\varphi^{cri}[\boldsymbol{o}(t+1), \boldsymbol{g}(t), \hat{\boldsymbol{a}}(t+1)]$. The caching decision network is updated by maximizing the expected long-term reward of

$$J(\phi_b) = \mathbb{E}\{\varphi^{cri}[\boldsymbol{o_b}(t), \hat{\boldsymbol{a}}_{\boldsymbol{b}}(t), \boldsymbol{g}(t-1)] \big|_{\hat{\boldsymbol{a}}_{\boldsymbol{b}}(t)=\varphi_b(\boldsymbol{o_b}(t),\boldsymbol{g}(t-1);\phi_b)}\}, \quad (34)$$

when the actor b takes the action $\hat{\boldsymbol{a}}_{\boldsymbol{b}}$. Following the chain rule, the gradients of the caching decision network parameters are given by

$$\nabla_{\phi_b} J(\phi_b) = \mathbb{E}\{\nabla_{\hat{\boldsymbol{a}}_{\boldsymbol{b}}}\varphi^{cri}[\boldsymbol{o_b}(t), \hat{\boldsymbol{a}}_{\boldsymbol{b}}(t), \boldsymbol{g}(t-1)]\nabla_{\phi_b} \varphi_b[\boldsymbol{o_b}(t), \boldsymbol{g}(t-1); \phi_b]\big|_{\hat{\boldsymbol{a}}_{\boldsymbol{b}}(t)=\varphi_b(\boldsymbol{o_b}(t),\boldsymbol{g}(t-1)}\}. \quad (35)$$

The parameters of the caching decision network are updated by $\phi_b(t+1) = \phi_b(t) + \xi\nabla_{\phi_b} J(\phi_b)$, where ξ is the learning rate. The parameters of the communication model are updated by minimizing the loss:

$$L(\phi^{com}) = \mathbb{E}\left\{\left[y(t) - \varphi^{cri}\big(\boldsymbol{o}(t), \boldsymbol{g}(t-1)\big)\right]^2\big|_{\boldsymbol{g}(t-1)=\varphi^{com}[\boldsymbol{o}(t-1),\hat{\boldsymbol{a}}(t-1),\boldsymbol{g}(t-2)]}\right\} - \mathbb{E}\left\{\varphi^{cri}\big[\boldsymbol{o}(t), \boldsymbol{g}(t-1)\big]\big|_{\boldsymbol{g}(t-1)=\varphi^{com}[\boldsymbol{o}(t-1),\hat{\boldsymbol{a}}(t-1),\boldsymbol{g}(t-2)]}\right\}. \quad (36)$$

The softmax layer of the actor networks ensures that the sum of all elements of the vector $\hat{\boldsymbol{a}}_{\boldsymbol{b}}$ is equal to 1. Naturally, the constraint with Eq.(2) is satisfied. However, in practice, any BS b may decide to cache over n_f segments for a single content f, i.e. $a_{b,f} > n_f$. The constraint associated with Eq.(1) is violated. Hence, we propose an action adjustment algorithm, which is listed in **Algorithm 1**. We first divide the elements of the vector $\hat{\boldsymbol{a}}_{\boldsymbol{b}}$ into two parts: the legitimate actions $\mathcal{I}$ and the illegitimate actions $\mathcal{K}$. Then, the total storage exceeding the limit is denoted by $\Delta = \sum_{k\in\mathcal{K}} \hat{a}_{b,k}N$. We assign the storage Δ equally to the contents having a legitimate action (line 3 to line 12). Finally, the adjusted action is given by $\boldsymbol{a}_{\boldsymbol{b}} = \lfloor \hat{\boldsymbol{a}}_{\boldsymbol{b}} N \rfloor$ according to (30).

Algorithm 1 Caching Action Adjustment.

Input: The output vector of the actor network b: $\hat{\boldsymbol{a}}_{\boldsymbol{b}}$;
Output: The adjusted action for the BS b: $\boldsymbol{a}_{\boldsymbol{b}}$;

1: Divide the elements of vector $\hat{\boldsymbol{a}}_{\boldsymbol{b}}$ into two parts:
   $\mathcal{I} = \{i \,|\hat{a}_{b,i}N \le n_i,\ \forall i \in \mathcal{F}\}$
   $\mathcal{K} = \{k \,|\hat{a}_{b,k}N > n_k,\ \forall k \in \mathcal{F}\}$
2: $\Delta = \sum_{k\in B} \hat{a}_{b,k}N$
3: **while** $\Delta > 0$ **do**
4: $\quad \mu = \frac{\Delta}{|\mathcal{I}|}$, where $|\mathcal{I}|$ is the size of set $\mathcal{I}$
5: $\quad$ **for** all $i \in \mathcal{I}$ **do**
6: $\quad\quad$ **if** $\mu > n_i - \hat{a}_{b,i}N$ **then**
7: $\quad\quad\quad$ Remove i from the set $\mathcal{I}$
8: $\quad\quad$ **end if**
9: $\quad\quad \hat{a}_{b,i} = \min\{n_i, \hat{a}_{b,i}N + \mu\}$
10: $\quad\quad \Delta = \Delta - \min\{\mu, n_i - \hat{a}_{b,i}N\}$
11: $\quad$ **end for**
12: **end while**
13: $\boldsymbol{a}_{\boldsymbol{b}} = \lfloor \hat{\boldsymbol{a}}_{\boldsymbol{b}} N \rfloor$

Given the above, the training process of the proposed multi-agent actor-critic network used for cooperative edge caching is summarized in **Algorithm 2**. Firstly, given the historical user request traces, we can train a VRNN model for content requests and a LSTM model for user access. Then, we generate E sample tracks, each of which has T samples (line 3 to line 16). Lastly, we update the parameters $\phi^{decs}, \phi^{cri}, \phi^{com}$ of the neural networks based on the experience pool replay (line 17 to line 28). During the online execution phase, the NGN core network employs a well-trained communication network and each BS employs a well-trained actor network, which carries out the optimal action based on the observations.

E. Convergence and Complexity

In order to analyze the convergence of distributed learning, we re-formulate the content placement algorithm of our edge caching system as a strategic game. Each agent is a game player, who independently takes an action before knowing the actions taken by the other players. After taking an action $\boldsymbol{a}$, each player b will obtain a payoff R_b. As defined in (14), the payoff of each player b is a function of the combinations of actions carried out by all the players instead of its own action $\boldsymbol{a}_{\boldsymbol{b}}$ only, because the number of segments transmitted from BS k to the BS b, i.e. r_b^k ($k \in \mathcal{B}, k \ne b$), is determined by the cache placements of other BSs. The action space of each player is a closed bounded set and the payoff function is continuous. Since the payoff is a linear function of the action, the payoff function is strictly quasi-concave. According to Debreu's theorem [36], these conditions ensure that a pure strategy Nash equilibrium exists. The system will converge to a stable state, where no single player can take a unilaterally action to achieve an increased payoff, which is a manifestation of the Nash equilibrium. Therefore, the existence of Nash equilibrium theoretically ensures the convergence of the proposed distributed learning algorithm.

The time complexity of the offline training process is proportional to the amount of training data as well as the

Algorithm 2 Training of the Multi-Agent Actor-Critic Model.

Require: Given the traces of the historical content request $\mathcal{T}_r$ and the traces of the historical user access $\mathcal{T}_a$;

Initialization: Initialize the parameters of the neural networks $\phi^{decs}, \phi^{cri}, \phi^{com}$; the target network $\phi^{tar} = \phi^{cri}$; the replay buffer D; exploration coefficient ε;

Train VRNN model for content request prediction with $\mathcal{T}_r$
Train LSTM model for user access prediction with $\mathcal{T}_a$
for each training step **do**
  for episode= 1 to E **do**
    Initialize $\boldsymbol{g}(0)$ and let $t = 1$;
    while $t < T$ **do**
      Generate an observation $\boldsymbol{o}_t$ based on $\mathcal{T}_r$ and $\mathcal{T}_a$;
      For each agent b, the actor network output an action vector $\hat{\boldsymbol{a}}_{\boldsymbol{b}}(t) = \varphi_b(\boldsymbol{o}_{\boldsymbol{b}}(t), \boldsymbol{g}(t-1))$;
      Action exploration $\hat{\boldsymbol{a}}_{\boldsymbol{b}}(t) = \hat{\boldsymbol{a}}_{\boldsymbol{b}}(t) + \varepsilon$;
      Obtain adjusted action $\boldsymbol{a}_{\boldsymbol{b}}(t)$ using **Algorithm 1**;
      Execute action $\boldsymbol{a}_{\boldsymbol{b}}(t)$ and get reward $R_b(t)$ and the subsequent observation $\boldsymbol{o}_b(t+1)$;
      $\boldsymbol{g}(t) = \varphi^{com}(\boldsymbol{o}(t), \hat{\boldsymbol{a}}(t), \boldsymbol{g}(t-1))$;
      $t = t + 1$;
    end while
    Store episode $\{\boldsymbol{g}(0), \boldsymbol{o}(1), \boldsymbol{a}(1), R(1); \cdots ; \boldsymbol{g}(T-1), \boldsymbol{o}(T), \boldsymbol{a}(T), R(T);\}$ in D;
  end for
  Randomly sample a minibatch of episodes M from D;
  for each episode in M **do**
    for $t = T$ down to 1 **do**
      Update the ϕ^{cri} by minimizing the loss (33);
      for all agents $b \in \mathcal{B}$ **do**
        Update ϕ^{decs} by maximizing loss (34);
      end for
      Update ϕ^{com} by minimizing loss (36);
    end for
    Update ϕ^{tar} by $\phi^{tar} = \tau\phi^{cri} + (1-\tau)\,\phi^{tar}$;
  end for
end for

training time, and thus we only focus on the running process. The time complexity of the running process is jointly decided by the structure of neural networks and the scales of both the state space and action space. As depicted in Fig. 3, the running process includes B actor networks and a communication network.

Since the input of the communication network includes the actions and observations of all agents, the input layer consists of $B \cdot (4F + 2)$ neurons. Let us assume that the number of memory blocks in the communication network is L_1 and the number of hidden units is H_1. A memory block contains 4 gates, each of which has a complexity of $O(H_1[B(4F+2)+H_1])$. Then, the total time complexity of our communication network is $O(L_1H_1BF)$, since the size of the input layer is usually larger than that of the hidden layer, i.e. $BF \gg H_1$.

The actor network includes three modules: access estimation, request estimation and caching decision, as shown in Fig. 4. The access estimation relies on an LSTM network, the input of which is the two-dimensional access state. Let us assume that the number of memory block is L_2 and the number of hidden units is H_2. Thus, estimating the access state has a complexity of $O(L_2H_2^2)$. The request estimation is modelled by a VRNN network, which consists of a VAE and a RNN. The input of VAE is the $2F$-dimensional request state. Let L_3 and H_3 denote the number of hidden layers and the number of neurons in each layer, respectively. The VAE has a complexity of $O(2FH_3 + H_3^2L_3)$. The input of the RNN is the request state and the latent variable, as seen in (19). Here, we assume that the RNN consists of L_4 cells, each of which includes H_4 hidden units. The RNN has a complexity of $O(L_4H_4(2F+H_4))$. The inputs of the decision network are the local observation and the global state. Let us assume that the number of hidden layers is L_5 and the number of neurons in each layer is H_5. The decision network has a complexity of $O([(2F+G)H_5 + H_5^2L_5])$.

In summary, the total time complexity of the running process is determined by the sum of a communication module and B actor modules, i.e. $O(B(L_1H_1F + L_2H_2^2 + 2FH_3 + H_3^2L_3 + 2L_4H_4F + L_4H_4H_4 + 2FH_5 + GH_5 + H_5^2L_5))$.

IV. Performance Evaluation

In this section, we evaluate the performance of the proposed multi-agent deep reinforcement learning (MADRL) based cooperative edge caching algorithm. For performance comparisons, we also implement the following algorithms as benchmarks:

- *Least recently used* (LRU): The LRU algorithm [37] will replace the cached contents, which are requested least frequently during the recent time slot. Each BS runs a LRU independently for its own content replacement. We assume that the LRU can replace 10 segments for a single content at a time.
- *Deep reinforcement learning* (DRL): The DRL algorithm [22] is used for deciding the cache placement of a single BS. Each BS employs a DDPG network and makes its own caching decisions independently based on the local observations. In order to accommodate the distributed edge caching scenario, the state, actions and rewards of the DRL model are defined to be the same as that of the actor network of Subsection II-B.
- *Multi-agent actor-critic* (MAAC): In MAAC (i.e. the decentralized caching algorithm proposed in [25]), the actor network of each BS is used for deciding the edge cache placement based solely on the local observations, and the critic network is used for estimating the overall reward of the actions. Notably, in contrast to the cooperative multi-agent actor-critic model proposed in this paper, there is no communication between agents and thus the inputs of the actor network do not include the global state. Furthermore, the actor network is merely a fully connected neural network operating without considering the temporal dynamics of content request and user access.

A. Simulation Setup

In order to simulate the random distribution of the UEs across the coverage area of ultra-dense cellular networks, we

TABLE III
PARAMETERS SETTINGS

Modules		Network Structure	Parameters
Actor	Content requests prediction	Prior: 3 fully connected layers Generation: 5 fully connected layers Recurrence: 5 LSTM layers Inference: 6 fully connected layers	Mini-batch size: 8 Number of epochs: 200 Sequence length: 5 Learning rate: 0.001 Weight decay coefficient: 0.0001 Experience replay buffer size: 50000 Discount factor: 0.9 Initial exploration coefficient: 0.03 Exploration fading factor: 0.9 Target network update rate: 0.01
	Access state prediction	6 LSTM layers	
	Cooperative caching decision	3 fully connected layers 1 softmax layer	
Critic		3 fully connected layers	
Communication		5 LSTM layers	

run the algorithms over the Geolife dataset [35], which keeps track of the real movement trajectories of mobile users. In the simulations, we divide the geographic area into 77 non-overlapping hexagons having the same side length, each of which is regarded as the coverage area of a BS, i.e. $B = 77$.

A user is associated with a specific BS when his/her moving trajectory of movement passes the coverage area of the BS. Due to the users' random movements, the number of access users in the various BSs' coverage area exhibits geographic differences.

In a real scenario, the content popularity would evolve over time. In order to simulate the temporal variation of the request rate for a certain content, the shot noise traffic model proposed in [38] is adopted in our experiments. Specifically, the request process for content f is assumed to be a time-inhomogeneous Poisson process. The instantaneous request rate is thus given by $V_f \cdot \lambda_f(t-\tau_f)$, where V_f is the average request rate during the lifespan of the content f, λ_f is the power law distribution, τ_f is the time instant when the content f becomes available to the users and τ_f is Poisson distributed. In this way, we can generate a time-varying content popularity.

Given the above model, we can generate a series of different request sequences for different contents by changing the model parameters. Mapping a generated content request to a trajectory in the Geolife dataset can produce a geographically distributed request. Naturally, the content requests in our simulations exhibit spatio-temporal dynamics.

In the simulations, we produce 40,000 requests in an episode. There are a total of 100 contents available to the users, i.e. $F = 100$. Each content is encoded by a Raptor code into different number of segments, ranging from 100 to 1000 segments. We assume that each segment has the same size. The cache capacity of each BS is set to 100 segments, i.e., $N = 100$. The cost of delivering a segment from the local cache of a BS is set to 1, i.e., $\alpha_b = 1$, $\forall b \in \mathcal{B}$. The cost of delivering a segment from BS k to BS b is set to 10, i.e., $\beta_b^k = 10$, $\forall k, b \in \mathcal{B}$. The cost of fetching a segment from the content server is set to 50, i.e., $\theta_b = 1$, $\forall b \in \mathcal{B}$. The cost of replacing a segment is set to 50, i.e., $\delta_b = 50$, $\forall b \in \mathcal{B}$.

In reality, different locations of the content may lead to different download speed. Fetching a content file from the local cache has the highest speed, which is randomly sampled from the range from 0.6 to 1 segments per second. The speed of fetching a content file from the nearby BS varies from 0.3 to 0.5 segments per second, while fetching it from the content server suffers from the slowest speed, which varies from 0.1 to 0.3 segments per second. The time duration of a slot is set to 10 minutes and the training episode is set to 200 slots. The parameter settings of the actor network, critic network and communication network are detailed in **Table III**. The training process and the testing process consist of 2,000 episodes and 100 episodes, respectively. In the simulations, we run the reference algorithms using the same request traces. All the simulations are conducted in a computer with an Intel(R) Xeon(R) Gold 5120 CPU and a Nvidia Tesla P4 GPU. During the simulations, we observe that each run takes 0.45s and occupies about 2Gb of physical memory. The time spent training the MADRL model to achieve convergence is about 30 hours.

B. Performance Metrics

We define three metrics to evaluate the performance:

- *The caching reward* quantifies the total long-term reward obtained from edge caching, which is defined as the sum of the immediate rewards for all BSs, i.e., $\tilde{R} = \sum_{t=1}^{T} R(t)$, where $R(t)$ is defined in Eq. (14) and T denotes the time duration of an episode.
- *The cache hit rate* quantifies the utilization of the edge caches, defined as the proportion of cached segments that have been requested by the users in an episode duration, which is represented by:

$$\tilde{H} = \frac{1}{T}\sum_{t=1}^{T}\sum_{b=1}^{B}\frac{I_b(n,t)}{NB}, \tag{37}$$

  where $I(\cdot)$ is an indicator. If the segment n cached in BS b is requested by the users during slot t, then $I(\cdot) = 1$, otherwise $I(\cdot) = 0$.
- *The traffic load* characterizes how the user requests are accommodated. Since the requested segments can be fetched either from the local BS, or from the nearby BSs, or alternatively from the content servers, we can define their traffic loads as the proportion of requested segments accommodated by each of them:

$$\begin{aligned}
\tilde{U}_{local} &= \frac{1}{T}\sum_{t=1}^{T}\frac{\sum_{b\in\mathcal{B}} r_b^b}{S(t)},\\
\tilde{U}_{nearby} &= \frac{1}{T}\sum_{t=1}^{T}\frac{\sum_{b\in\mathcal{B}}\sum_{k\in\mathcal{B},k\neq b} r_b^k}{S(t)},\\
\tilde{U}_{server} &= 1 - \tilde{U}_{local} - \tilde{U}_{nearby},
\end{aligned} \tag{38}$$

  where $S(t)$ is the total number of segments requested by the users during slot t and r_b^k denotes the segments fetched from BS k to BS b, as defined in Subsection II-B.

C. Results Analysis

The learning process of the MADRL is shown in Fig. 7. The loss decays rapidly up to about 1,200 episodes and

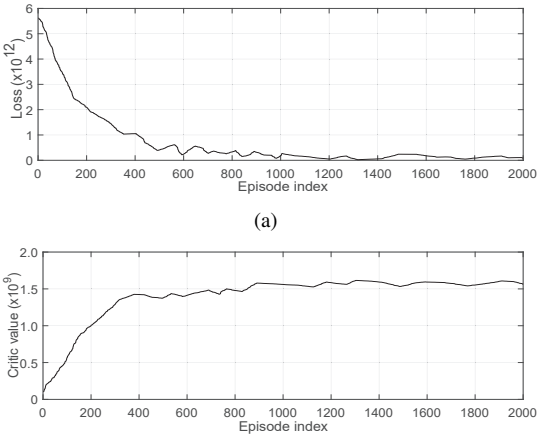


Fig. 7. Learning process of the MADRL algorithm. (a) The loss of critic network. (b) The output value of critic network.

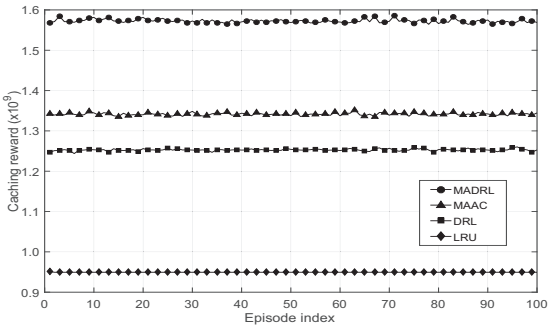


Fig. 8. The caching reward in each episode.

then gradually stabilizes. Since the loss represents the error between the critic value as defined in (32) and the actual reward, the critic value is approaching the actual reward as the training goes on, which is in accordance with the variation of the critic value curve. This indicates that the learning algorithm converges after about 1,200 episodes' training and then the well-trained critic network can be used for accurately estimating the value function.

Fig. 8 shows the caching reward obtained by the four algorithms in each episode during the testing phase. Explicitly, the MADRL attains the highest reward, followed by the MAAC and the DRL, and finally the LRU. This indicates that the MADRL can readily coordinate the BSs to take full advantage of the distributed edge caches and thus achieves a higher reward for all BSs. Compared to the MADRL, the MAAC also adopts the multi-agent actor-critic framework, but there is no communication among the agents. Specifically, for MAAC, each BS makes its caching decisions purely based on the local observations without considering the actions and states of the nearby BSs. However, the caching reward of each BS, as defined in (14), depends not only on its own cache placement but also on the other BSs' cache placements. Furthermore, the MAAC uses a simple fully connected neural network as an actor, which cannot explicitly model the variability of the observed states. Accordingly, the actor network of the MAAC cannot estimate the optimal strategy function. Upon comparing

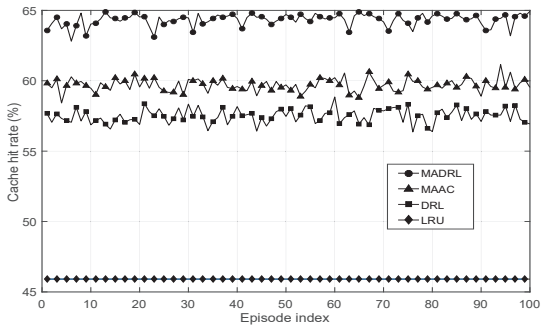


Fig. 9. The average hit rate of the edge caches.

with the MADRL, we find that the cooperations among the edge caches can significantly improve the reward, because the decision making of each BS is entirely independent in DRL. In Fig. 8, the learning based algorithms, including MADRL, DRL and MAAC, outperform the LRU which uses heuristic methods for cache replacement. We also find that the reward of the LRU remains unchanged vs. the episode index, while the rewards of the other algorithms vary slightly, although the request traces used in the testing are the same between episodes. The reason for this trend is that the learning based algorithms use the ε-greedy policy to explore the new actions, which may result in small perturbations to the rewards.

We plot the average hit rate of the edge caches in Fig. 9. The MADRL achieves the highest cache hit rate, which implies the best utilization of the edge caches. Although the storage of a single BS is quite limited, the MADRL can make the BSs work cooperatively by sharing the cached contents among the BSs, so that the users can fetch the contents both from the local and from nearby BSs rather than from the content servers. The collaboration among BSs avoids caching the same contents redundantly in different BSs, which saves the edge storages. By contrast, the DRL and the LRU make their caching decisions independently, and thus the BSs may cache duplicate contents. They may cache much less contents in the limited edge storage than the MADRL. As a result, based on the same request traces, the requested contents are more likely to activate the edge caches using MADRL than upon using the DRL and LRU. Hence, the MADRL has a much higher cache hit rate than the DRL and the LRU. Meanwhile, since the deep learning model characterizes the variation of content popularity better than the heuristic model, the DRL achieves a higher cache hit rate than the LRU. In Fig. 9, we also find that the MAAC achieves a slightly lower cache hit rate than the MADRL. For MAAC, the decision making of each BS does not consider other BSs' cache placement. As a result, a BS may cache a popular content even if this content can be widely found in the nearby BSs. The redundant caching may waste the limited storage, which is supposed to cache more contents so as to further improve the cache hit rate.

In order to characterize how the users' requests are accommodated, in Fig. 10, we plot the proportions of the requested segments that respectively are transmitted from the local BS, from the nearby BS and from the content server. Observe from Fig. 10(a) that only about 21% of the requested segments are

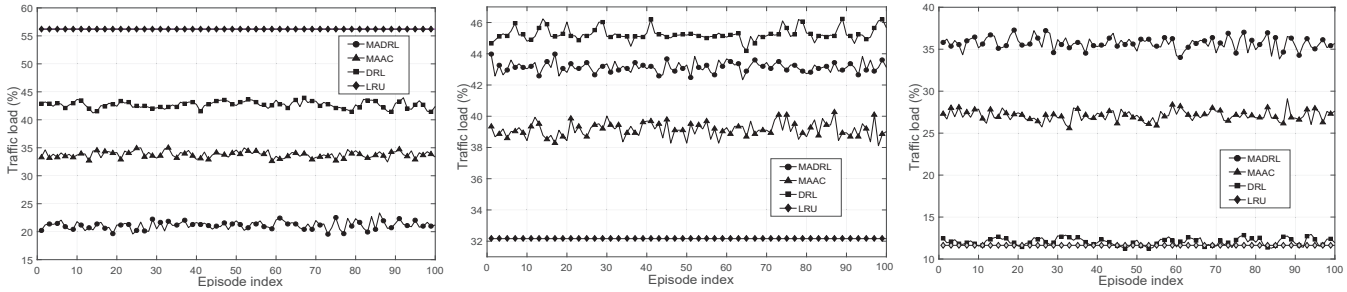


Fig. 10. The proportion of requested segments accommodated by (a) the content servers, (b) the local BS and (c) the nearby BSs.

transmitted from the content servers using the MADRL, which is far below the proportions of 56%, 43% and 34% upon using the other algorithms. This result implies that the MADRL can make the best use of the edge caches for reducing the content retrieval through the core networks, which is in accordance with the results of Fig. 8, since the cost of fetching the content from the servers is higher than that from the edge caches. Upon comparing Fig. 10(b) to (c), we find that although the MADRL (43%) has a slightly lower proportion of segments fetched from the local BS than the DRL (45%), the proportion of segments fetched from the nearby BSs using the MADRL (36%) is much higher than that using the DRL (12%). In DRL, each BS only considers accommodating the requests from its local users while deciding the cache placement. By contrast, the MADRL also considers the cooperation among the BSs. Specifically, for a hot spot, the limited storage of the local BS is not enough for caching all the popular contents, but the nearby BSs can help to accommodate some requests by content sharing. This result indicates that the MADRL mitigates the traffic burdens imposed on the backhauls.

Fig. 11 shows the performance under different cache sizes, which range from 50 to 300 segments with a step size of 50 segments. Observe in Fig. 11(a) that the caching reward increases upon increasing the BS's cache size. Larger storage capacities of the edge caches enable us to cache more contents. Hence the users' requests are more likely to be accommodated by the edge caches, which significantly alleviates the traffic burden imposed both on the backhaul and on the core networks. The reduction of network resources also means the increase of caching reward. It is also seen in Fig. 11(b) that the cache hit rate decreases upon increasing the BS's cache size. A small cache size only allows us to retain the most popular contents, thus the cached segments are more likely to be requested by the users, which results in a high cache hit rate. By contrast, a large cache size would reduce the likelihood that the cached contents are eventually activated. We also find that the performance gaps between the MADRL and the other algorithms become wider as the cache size increases. The BS having a larger cache size can retain more contents for other "hot" BSs, which naturally promotes the content sharing among the BSs.

A further experiment is conducted for evaluating the performance under different numbers of content requests, which are set to span from 20,000 to 120,000 in an episode. We generate

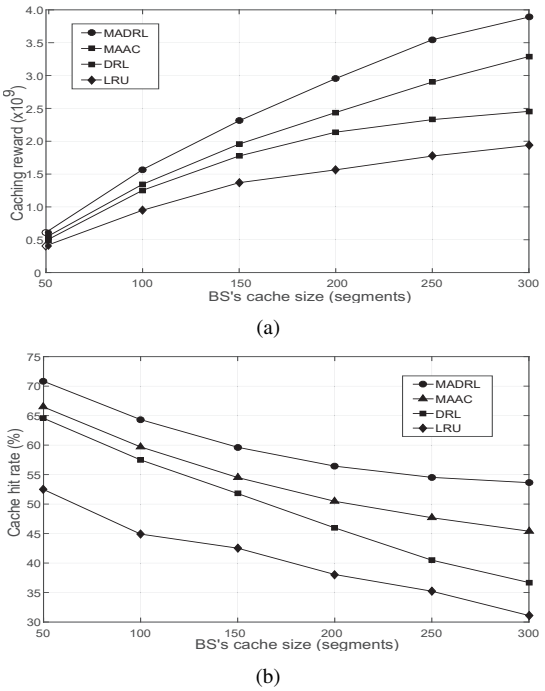
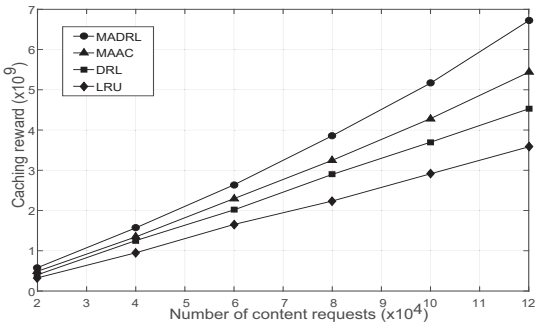


Fig. 11. (a) The caching reward and (b) the cache hit rate under different cache sizes.

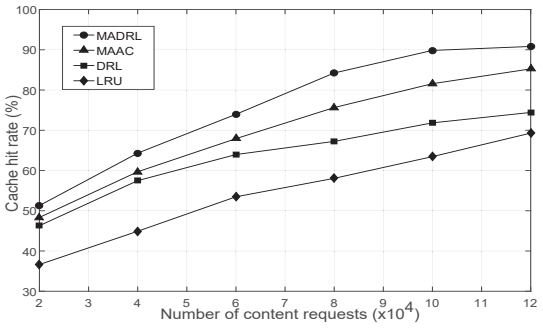
the request traces by scaling up the average request rate for each content, while keeping the content popularity distribution the same. As shown in Fig. 12(a), the caching reward increases with the number of contents requests and the performance gaps among these algorithms become wider. Since the caching reward characterizes the cumulative cost savings for all requests, the increase of requests may lead to the increase of cost saving. In Fig. 12(b), the cache hit rate also increases with the number of contents requests. Since the generation of contents requests obeys the popularity distribution, the requests for the contents having a low popularity are more likely to occur as the number of sample requests increases.

V. Conclusions and Future Works

In this paper, we formulated the cooperative caching problem as a POMDP-based multi-agent decision problem, which jointly optimized the costs of fetching contents from the local BS, the nearby BSs and the content servers. To solve this problem, we devised a multi-agent actor-critic framework,



(a)



(b)

Fig. 12. (a) The caching reward and (b) the cache hit rate under different number of content requests.

where a communication module was introduced to extract and share the variations of the global states among the BSs. Since the full information of the state is explicitly unkonwn, we proposed a VRNN-based content request estimation model and an LSTM-based user access estimation model. Accordingly, we proposed a multi-agent deep reinforcement learning based cooperative edge caching algorithm, where each BS made its own caching decisions based on both the local and the global states. Our experimental results verified that the proposed algorithm facilitated the collaboration between the edge caches, and thus reduces the traffic loads on the backhaul links, while improving the edge cache hit rate.

REFERENCES

[1] 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.

[2] Cisco, "Cisco visual networking index: Forecast and trends, 2017-2022 white paper," 2019. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html

[3] J. Yang, B. Yang, S. Chen, Y. Zhang, Y. Zhang, and L. Hanzo, "Dynamic resource allocation for streaming scalable videos in SDN-aided dense small-cell networks," *IEEE Trans. Commun.*, vol. 67, no. 3, pp. 2114–2129, 2018.

[4] 3GPP, "NR and NG-RAN overall description," 3GPP Technical Specification TS 38.300, Release 15, 2019.

[5] C. Pan, M. Elkashlan, J. Wang, J. Yuan, and L. Hanzo, "User-centric C-RAN architecture for ultra-dense 5G networks: Challenges and methodologies," *IEEE Commun. Mag.*, vol. 56, no. 6, pp. 14–20, 2018.

[6] L. Qiu and G. Cao, "Popularity-aware caching increases the capacity of wireless networks," *IEEE Trans. Mobile Comput.*, vol. 19, no. 1, pp. 173–187, 2019.

[7] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, 2017.

[8] J. Li, H. Chen, Y. Chen, Z. Lin, B. Vucetic, and L. Hanzo, "Pricing and resource allocation via game theory for a small-cell video caching system," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 8, pp. 2115–2129, 2016.

[9] G. Paschos, E. Bastug, I. Land, G. Caire, and M. Debbah, "Wireless caching: Technical misconceptions and business barriers," *IEEE Commun. Mag.*, vol. 54, no. 8, pp. 16–22, 2016.

[10] W. Wen, Y. Cui, F.-C. Zheng, S. Jin, and Y. Jiang, "Random caching based cooperative transmission in heterogeneous wireless networks," *IEEE Trans. Commun.*, vol. 66, no. 7, pp. 2809–2825, 2018.

[11] C. Liang, Y. He, F. R. Yu, and N. Zhao, "Enhancing QoE-aware wireless edge caching with software-defined wireless networks," *IEEE Trans. Wireless Commun.*, vol. 16, no. 10, pp. 6912–6925, 2017.

[12] L. T. Tan, R. Q. Hu, and L. Hanzo, "Heterogeneous networks relying on full-duplex relays and mobility-aware probabilistic caching," *IEEE Trans. Commun.*, vol. 67, no. 7, pp. 5037–5052, 2019.

[13] M. Gregori, J. Gómez-Vilardebó, J. Matamoros, and D. Gündüz, "Wireless content caching for small cell and D2D networks," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 5, pp. 1222–1234, 2016.

[14] J. Hachem, N. Karamchandani, and S. Diggavi, "Content caching and delivery over heterogeneous wireless networks," in *Proc. INFOCOM*. IEEE, 2015, pp. 756–764.

[15] W. Jiang, G. Feng, and S. Qin, "Optimal cooperative content caching and delivery policy for heterogeneous cellular networks," *IEEE Trans. Mobile Comput.*, vol. 16, no. 5, pp. 1382–1393, 2016.

[16] B. 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, 2016.

[17] J. Kwak, Y. Kim, L. B. Le, and S. Chong, "Hybrid content caching in 5G wireless networks: Cloud versus edge caching," *IEEE Trans. Wireless Commun.*, vol. 17, no. 5, pp. 3030–3045, 2018.

[18] J. Kwak, L. B. Le, H. Kim, and X. Wang, "Two time-scale edge caching and BS association for power-delay tradeoff in multi-cell networks," *IEEE Trans. Commun.*, vol. 67, no. 8, pp. 5506–5519, 2019.

[19] M. Chen, W. Saad, and C. Yin, "Echo state networks for proactive caching in cloud-based radio access networks with mobile users," *IEEE Trans. Wireless Commun.*, vol. 16, no. 6, pp. 3520–3535, 2017.

[20] A. Sadeghi, F. Sheikholeslami, and G. B. Giannakis, "Optimal and scalable caching for 5G using reinforcement learning of space-time popularities," *IEEE J. Sel. Topics Signal Process.*, vol. 12, no. 1, pp. 180–190, 2017.

[21] Y. He, N. Zhao, and H. Yin, "Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach," *IEEE Trans. Veh. Technol.*, vol. 67, no. 1, pp. 44–55, 2018.

[22] L. T. Tan and R. Q. Hu, "Mobility-aware edge caching and computing in vehicle networks: A deep reinforcement learning," *IEEE Trans. Veh. Technol.*, vol. 67, no. 11, pp. 10 190–10 203, 2018.

[23] G. Qiao and S. Leng, "Deep reinforcement learning for cooperative content caching in vehicular edge computing and networks," *IEEE Internet Things J.*, vol. 7, no. 1, pp. 247–257, 2020.

[24] W. Jiang and G. Feng, "Multi-agent reinforcement learning for efficient content caching in mobile D2D networks," *IEEE Trans. Wireless Commun.*, vol. 18, no. 3, pp. 1610–1622, 2019.

[25] C. Zhong, M. C. Gursoy, and S. Velipasalar, "Deep reinforcement learning based edge caching in wireless networks," *IEEE Trans. Cognitive Commun. and Netw.*, 2020, DOI 10.1109/TCCN.2020.2968326.

[26] J. Song, M. Sheng, T. Q. Quek, C. Xu, and X. Wang, "Learning-based content caching and sharing for wireless networks," *IEEE Trans. Commun.*, vol. 65, no. 10, pp. 4309–4324, 2017.

[27] A. Shokrollahi, "Raptor codes," *IEEE Trans. Inf. Theory*, vol. 52, no. 6, pp. 2551–2567, Jun. 2006.

[28] X. Jiang, X. Wang, H. Xi, and F. Liu, "Centralized optimization for Dec-POMDPs under the expected average reward criterion," *IEEE Trans. Autom. Control*, vol. 62, no. 11, pp. 6032–6038, 2017.

[29] J. Chung, K. Kastner, L. Dinh, K. Goel, A. C. Courville, and Y. Bengio, "A recurrent latent variable model for sequential data," in *Advances in neural information processing systems*, 2015, pp. 2980–2988.

[30] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," in *9th International Conf. on Artificial Neural Networks (ICANN)*. IET, 1999.

[31] M. Chen, Y. Hao, L. Hu, K. Huang, and V. K. Lau, "Green and mobility-aware caching in 5G networks," *IEEE Trans. Wireless Commun.*, vol. 16, no. 12, pp. 8347–8361, 2017.

[32] C. Li, H. J. Yang, F. Sun, J. M. Cioffi, and L. Yang, "Multiuser overhearing for cooperative two-way multiantenna relays," *IEEE Trans. Veh. Technol.*, vol. 65, no. 5, pp. 3796–3802, 2016.

[33] C. Li, S. Zhang, P. Liu, F. Sun, J. M. Cioffi, and L. Yang, "Overhearing protocol design exploiting inter-cell interference in cooperative green-networks," *IEEE Trans. Veh. Technol.*, vol. 65, no. 1, pp. 441–446, 2016.
[34] Y. Pu, Z. Gan, R. Henao, X. Yuan, C. Li, A. Stevens, and L. Carin, "Variational autoencoder for deep learning of images, labels and captions," in *Proc. NIPS 2016*, 2016, pp. 2352–2360.
[35] Y. Zheng, X. Xie, and W. Ma, "Geolife: A collaborative social networking service among user, location and trajectory." *IEEE Data Eng. Bull.*, vol. 33, no. 2, pp. 32–39, 2010.
[36] D. Fudenberg and J. Tirole, *Game Theory*. MIT Press, 1996.
[37] D. Lee and J. Choi, "LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies," *IEEE Trans. Comput.*, no. 12, pp. 1352–1361, 2001.
[38] S. Traverso and M. Ahmed, "Temporal locality in today's content caching: why it matters and how to model it," *ACM SIGCOMM*, vol. 43, no. 5, pp. 5–12, 2013.

Shuangwu Chen received the B.S. and Ph.D. degrees from University of Science and Technology of China (USTC), Hefei, China, in 2011 and 2016, respectively. He is currently an associate professor at USTC. His research interests include future network, multimedia communication and stochastic optimization.

Zhen Yao received the B.S. and Ph.D. degree from University of Science and Technology of China (USTC) in 2015 and 2020 . His research interests include software defined networking and in-network intelligence assisted by machine learning.

Xiaofeng Jiang (M'13) received the B.E. and Ph.D. in information science and technology from University of Science and Technology of China (USTC), Hefei, China, in 2008 and 2013. He is currently an associate professor in the School of Information Science and Technology, USTC. His recent research interests include discrete event dynamic system, tensor analysis and big data, future network and cognitive communications.

Jian Yang received the B.S. and Ph.D. degrees from the University of Science and Technology of China (USTC), Hefei, China, in 2001 and 2006, respectively. From 2006 to 2008, he was a postdoctoral scholar in the Department of Electronic Engineering and Information Science in USTC. Since 2008 he has been employed as an associate professor in the Department of Automation, USTC.

He is currently a professor in the School of Information Science and Technology, USTC. His research interests include future network, distributed system design, modeling & optimization, multimedia over wired & wireless and stochastic optimization.

Lajos Hanzo (http://www-mobile.ecs.soton.ac.uk,https://en.wikipedia.org/wiki/Lajos_Hanzo) (FIEEE'04) received his Master degree and Doctorate in 1976 and 1983, respectively from the Technical University (TU) of Budapest. He was also awarded the Doctor of Sciences (DSc) degree by the University of Southampton (2004) and Honorary Doctorates by the TU of Budapest (2009) and by the University of Edinburgh (2015). He is a Foreign Member of the Hungarian Academy of Sciences and a former Editor-in-Chief of the IEEE Press. He has served several terms as Governor of both IEEE ComSoc and of VTS. He has published 1900+ contributions at IEEE Xplore, 19 Wiley-IEEE Press books and has helped the fast-track career of 123 PhD students. Over 40 of them are Professors at various stages of their careers in academia and many of them are leading scientists in the wireless industry. He is also a Fellow of the Royal Academy of Engineering (FREng), of the IET and of EURASIP.