Toward Smart and Cooperative Edge Caching for 5G Networks: A Deep Learning Based Approach

Haitian Pang*†, Jiangchuan Liu†, Xiaoyi Fan†, Lifeng Sun*
* Tsinghua National Laboratory for Information Science and Technology, and
Department of Computer Science and Technology, Tsinghua University, Beijing, China
† School of Computer Science,Simon Fraser University, Burnaby, BC, Canada
Email: {pht14@mails., sunlf@}tsinghua.edu.cn, {jcliu, xiaoyif}@sfu.ca

***Abstract*—The emerging 5G mobile networking promises ultra-high network bandwidth and ultra-low communication latency (<1ms), benefiting a wide range of applications, including live video streaming, online gaming, virtual and augmented reality, and Vehicle-to-X, to name but a few. The backbone Internet, however, does not keep up, particularly in latency (>100ms), due to its store-and-forward design and the physical barrier from signal propagation speed, not to mention congestion that frequently happens. Caching is known to be effective to bridge the speed gap, which has become a critical component in the 5G deployment as well. Besides storage, 5G base stations (BSs) will also be powered with strong computing modules, offering *mobile edge computing* (MEC) capability. This paper explores the potentials of edge computing towards improving the cache performance, and we envision a learning-based framework that facilitates smart caching beyond simple frequency- and time-based replace strategies and cooperation among base stations. Within this framework, we develop *DeepCache*, a deep-learning-based solution to understand the request patterns in individual base stations and accordingly make intelligent cache decisions. Using mobile video, one of the most popular applications with high traffic demand, as a case, we further develop a cooperation strategy for nearby base stations to collectively serve user requests. Experimental results on real-world dataset show that using the collaborative DeepCache algorithm, the overall transmission delay is reduced by 14%∼22%, with a backhaul data traffic saving of 15%∼23%.**

***Index Terms*—5G Networks, Mobile Edge Computing, Collaborative Caching, Deep Learning.**

I. Introduction

The 5G wireless network is designed to increase the capacity of cellular networks by means of more bandwidth, larger scale of antennas, higher frequency reuse with network densification etc. However, the network requirements for applications in 5G networks is even increasing with higher speed. The global mobile data traffic has grown 63% in 2016 and is predicted to increase another seven-fold by 2021 [1], among which mobile video is predicted to account for 78% mobile data traffic. The quality of individual video streams is rapidly improving as well, with a majority of them have become 1080p high resolution (HD) or even 4K Ultra HD (UHD), posing significant challenges towards delivering high Quality of Service (QoS) to video consumers. In this way, we show the case study of mobile video caching in 5G networks. While other emerging applications also require high network performance: The Virtual Reality video streaming will grow 11-fold by 2021, which requires both high throughput (200Mbps∼) and low latency (∼10 ms). Self-driving vehicles will be equipped with sensing, processing, and communication abilities, and connected to the 5G network all the time. Vehicles will require a large amount of data from the network, e.g., transportation information, Augmented Reality navigation streaming, and in-car entertainment.

However, the backbone Internet does not keep up with the higher application requirements in latency and bandwidth, due to the store-and-forward design, the physical barrier from signal propagation speed, as well as the frequently happened congestion. Thus it is significant to envision a more sustainable roadmap towards future networks. To this end, mobile edge caching is of unprecedented importance in the 5G network to provide better QoS for the novel applications in two aspects: (1) Caching contents in the BSs can reduce the network latency significantly, and most 5G applications like VR/AR and V2X cannot bear the high delay in remote Internet delivery. (2) Caching contents in the edge can reduce the data traffic in the core network and save bandwidth for the Internet.

Edge caching in 5G networks is implemented by equipping BSs with MEC servers, which can provide not only storage capacity but also CPU/GPU computing for intelligent decision making. Each BS can collect the local content requests in real time, and then use the collected data to make intelligent decisions about cache replacement with the aid of powerful computing ability, including the support for deep learning.

In the 5G network, interconnected MEC servers enable edge caching to be implemented in a cooperative and intelligent way. Specifically, all the BSs form a mesh network via the wired network. Upon request, the desired content can be delivered with the short-range communications between the BSs and users directly. In this way, the transmission delay can be reduced significantly and the network bandwidth can be saved. The challenges for such cooperative edge caching system is how to deploy intelligent caching replacement algorithm and how to design an efficient cooperation mechanism among BSs.

Most of today's caching systems rely on such *rule-based* cache replacement algorithms as FIFO, Least Recently Used (LRU), Least Frequently Used (LFU), or their variants [2]–[4]. These algorithms follow simplified rules and are easy to

MEC server	Manufacturer	Storage (GB)	CPU	GPU
Jetson TX2	NVIDIA	32	Quad ARM A57	NVIDIA Pascal
HyperFlex Edge	Cisco	128	Intel Xeon	/
Edge Cloud	Intel	16~64	Intel Xeon	/
Power S822LC	IBM	32~1024	Power 8	NVIDIA Tesla P100

TABLE I: Examples of real-world MEC servers.

implement in reality, but the fixed rules can hardly adapt to the dynamic content access patterns. With the advancement in data analytics, *forecast-based* cache replacement algorithms have recently been suggested [5]. A well-trained model with proper feature engineering can achieve high hit ratio. Yet it heavily depends on the specific data for training and is hardly adaptive. A more intelligent cache replacement algorithm is expected for higher cache efficiency.

Many previous works focus on the cooperation mechanisms for the 5G wireless network, while they often refer to the literature for the cache placement algorithm. Some works modeled the content popularity as Zipf distribution and required popularity estimation methods, some other works adopted simple replacement algorithms, e.g., LRU, LFU as the basis. This makes the cooperation mechanism lightly coupled with the cache placement problem. In addition, once the cache placement algorithm makes wrong estimations or replacement decisions, the cooperation mechanism will achieve poor performance. In this way, a better solution is to use a joint framework to incorporate both the cache replacement and the corresponding cooperation mechanism.

In this paper, we propose DeepCache, an efficient deep-learning-based cache replacement strategy for the edge network, which learns the caching strategy automatically from the request sequence in real-time. Given the dynamics of video access patterns in both short- and long-terms, DeepCache employs a deep LSTM network [6], an advanced neural network architecture with memory to characterize the sequential pattern. Moreover, in order to apply DeepCache to the cooperative caching scenario, we introduce a cooperation mechanism for the DeepCache algorithm. With accurate content popularity predicted by DeepCache, we further consider the effect on the mobile edge network when replacing a content. We perform the experiment on a large scale real-world mobile video request dataset. Experiment results show that the proposed cooperative caching algorithm can reduce the transmission delay by 14%~22%, and save 15%~23% backhaul data traffic.

The remainder of the paper is organized as follows. The system architecture is introduced in Section II. The design of DeepCache and the corresponding cooperation mechanism are provided in Section III and IV, respectively. Experimental results are provided in Section V. We conclude the paper in Section VI.

II. System Architecture

We consider a general 5G network scenario, as depicted in Fig. 1. The base stations are equipped with MEC servers, which can be used to cache and deliver frequently requested

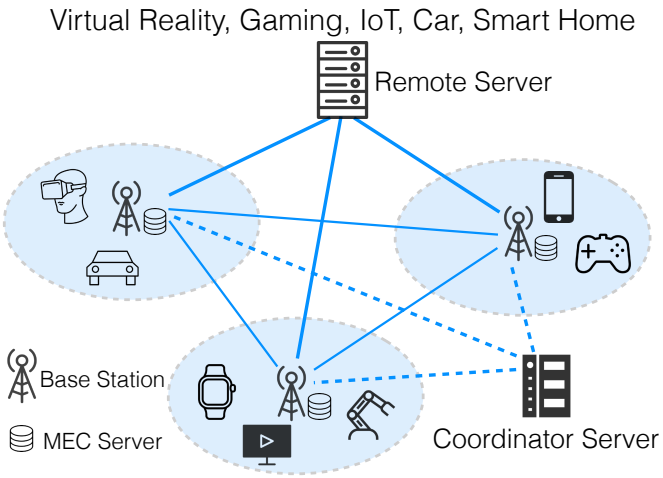


Fig. 1: Framework of Cooperative edge caching system.

contents. In order to further understand the MEC servers, we provide the parameters of some MEC servers as CPU, GPU, and storage in Table I. We find the storage sizes of the MEC servers range from 16GB to 1024GB, which can be used to cache popular contents and is believed to increase in the future. Along with sufficient cache space, the computing ability can satisfy intelligent deep learning computation as high-performance CPUs and GPUs are enabled. Especially, NVIDIA Jetson TX2 is an MEC server designed for artificial intelligence in the edge[1]. In this way, MEC servers make cooperative edge caching a reality as the powerful computation (including support for deep learning) and large storage size.

Applications like 4K streaming, virtual reality streaming, cloud gaming, vehicle communication, and robot arm initiate data requests via the BSs in the 5G network. If a request for an uncached content is initiated, the data is delivered from the remote server in the core network. Once the requested content is cached in the BS, the content can be delivered to the user from the local cache. In addition, if the requested content is not cached in the local BS, but cached in a nearby BS, the content can be delivered from the nearby BS. A coordinator server is employed to collect the information from the edge BSs, assign content retrieval routing, and decide the content eviction. Hitting local cache or nearby cache can reduce the transmission delay as well as the traffic over the network.

III. DeepCache: Deep Learning-based Cache Algorithm

We first introduce the deep neural network architecture, and then design an approximate method for the model training.

[1]https://developer.nvidia.com/embedded-computing

A. Deep LSTM Network for Caching

Inspired by the recent success of deep LSTM [6] network processing streaming data, we design a novel LSTM network algorithm for content caching. Despite a large number of neural network types in deep learning, we select the LSTM as the network unit as it has the potential to address the challenges for our problem: 1) the video content requests form a time sequence naturally, while LSTM is especially good at sequence modeling task. 2) LSTM's sophisticated network structure enable itself with strong representation ability from raw data input, thus requires little data pre-processing. 3) The memory structure inside LSTM can make full use of the historical sequence information when making decisions or predictions. 4) The LSTM network can be updated online to capture the timely popularity of the contents.

The architecture of the proposed network has multiple fully-connected LSTM layers, and one softmax layer which outputs the prediction result. Specifically, the LSTM layers take the historical sequence as input and output a vector $\mathbf{x}$ with dimension C. Then, we output the vector $\mathbf{x}$ to a $softmax$ layer to calculate the probability for each content to arrive. The softmax function is defined as follows:

$$\text{softmax}_i = \frac{e^{x_i}}{\sum_{i'=1}^{C} e^{x'_i}} \tag{1}$$

The full connection architecture enables the network to fully exploit the inherent correlations among cells, hence represent the complicated content request pattern better. In order to achieve better performance, we explore two hyperparameters in LSTM network: the number of hidden layers, and the number of cells in each layer, both of which have potential to influence the performance.

In the deep LSTM network, the content request sequence can be naturally considered as the feature to predict which content to evict when a content out of the local cache arrives. Specifically, each content is represented as one-hot encoding. We can train the model by feeding the "raw" request sequence to the deep LSTM model in an online fashion. It gradually learns to make better caching replacement decisions through the online training process, in which the algorithm try to achieve the optimal caching performance on the historical data.

B. Approximate Method for Model Training

Unlike most sequential learning tasks which predict the next-step instance with the highest probability, whose ground truth can be obtained when the next-step instance arrives, we aim to decide which content to evict from the cache space. Specifically, we want to find the content which is most *unlikely* to arrive among the cached contents. In this problem, challenges arise in two aspects: On one hand, it is hard to let the neural network output which content should be evicted directly, because this depends on the knowledge of which contents are cached currently. On the other hand, the ground truth for training cannot be obtained in the near future.

In order to meet the first challenge, we define caching priority as the benefit of keeping a content in the cache,

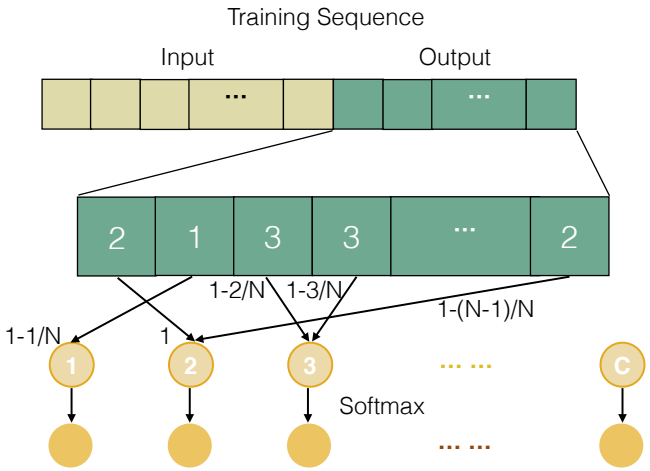


Fig. 2: Input and Output in the training phase.

and calculate the caching priority of all the contents with DeepCache. Then we rank the cached contents by the caching priority score and evict the content with the lowest score.

We design an approximate method to cope with the second challenge. The optimal strategy is to evict the content which ranks last in order in the future [7]. However, the optimal strategy relies on all the future information, which cannot be obtained in reality. The most commonly used one-hot encoding in prediction tasks is not applicable either, as it cannot reflect the priority of the cached contents. In this way, we design a method that uses a request sequence to approximate the ground truth. Fig. 2 shows the training phase, where $\{s_1, s_2, ..., s_M\}$ is the input sequence with length M, and $\{s_{M+1}, s_{M+2}, ..., s_{M+N}\}$ is the output sequence of length N used to generate the approximate caching priority, where s_i denotes the requested content. We pay particular attention to the output sequence. Inspired by the optimal strategy of content replacement, we notice that the order of the future contents is an important information. We first set the caching priority in the n-th order as $f(n)$, which is monotonously decreasing with n. This is consistent with the fact that a content arriving in the near future should be kept in the local cache. In addition, the number of a content arriving is another important factor, as a content being requested many times should be kept in the local cache. Incorporating the influence of both the order and the number of contents, the overall priority of content s is denoted as $W(s)$:

$$W(s) = \sum_{I(s_{M+n}=s)} f(n), \tag{2}$$

where $I(x) = 1$, when x is true. In summary, $W(s)$ is the sum weight of a content in all arrivals.

Generally speaking, the video request datasets can be divided into two categories. In one category, the popularity of video contents is highly time-variant, and the lifespan of a content is short. For example, video contents like weekly TV shows and TV series burst in views and die down soon. In the other one, the popularity of video contents is stable, and the lifespan of a content is long. For example, most movie

contents release long ago, and still attract the fans to watch. In order to characterize the patterns of different video contents, we provide a generalized formulation of $f(n)$ for all possible datasets as:

$$f(n) = 1 - (\frac{n-1}{N})^{\alpha},\ \alpha > 0, \tag{3}$$

where α is a parameter that is different among datasets, and N is the output length. In the example depicted in Fig. 2, we set $\alpha = 1$. The priority of content 1 is $1 - \frac{1}{N}$ as it arrives in the second order once. The priority of content 2 is $1 + \frac{1}{N}$ as it arrives in the first order ($f(1) = 1$) and the N-th order ($f(N) = \frac{1}{N}$), respectively. The formulation of $f(n)$ has the following properties: (1) it is monotonously decreasing with n, (2) the descent rate can be adjusted by α.

After deriving the weights of all contents, we employ a $softmax$ function with $W(s)$, $s \in \mathcal{S}$ as the input and the output probability distribution is denoted as $P(s)$, where $\mathcal{S}$ is the content library. We train the DeepCache model in a supervised learning framework, and define the loss function as the cross-entropy error:

$$loss = \sum_{s=1}^{S} P^g(s) \cdot \log(P(s)), \tag{4}$$

where $P^g(s)$ is the approximate priority, and $P(s)$ is the predicted priority of content s. We take the derivative of the loss function through back-propagation with respect to all parameters and update the parameters with the stochastic gradient descent.

IV. Cooperative DeepCache

The above mentioned DeepCache algorithm can achieve efficient caching performance in individual MEC-enabled BS. However, as all the BSs are deployed at the edge of the network and connected with each other via the wired network, contents can be delivered to a user from nearby BSs, which can reduce the network delay and traffic potentially. In this scenario, trying to seek the optimal performance on individual BS may not result in the optimal result of the mobile edge network. For example, if one content is popular and cached in many BSs, caching redundancy will happen, as the users can be served by nearby BSs with close quality. In this way, a cooperation mechanism among base stations is expected to solve the problem.

A. Problem Formulation

In this paper, we consider an edge network consisting of B cache-enabled base stations, denoted as $\mathcal{B} = \{1, 2, ..., B\}$. Additionally, we denote the remote server as $b = 0$, and all the potential servers (the remote server and the MEC servers) as $\mathcal{B}^* = \mathcal{B} \cup \{0\}$. The network delay between arbitrary two servers is measured and denoted as $C(b_i, b_j), \forall b_i, b_j \in \mathcal{B}^*$. When $b_i = b_j$, the network delay $C(b_i, b_j)$ is zero. When b_i and b_j are different servers, we need to measure the network delay. Note that there exist methods to measure and estimate the network delay between two servers [8], thus this is out of scope of this paper. The contents are denoted as $\mathcal{S} = \{1, 2, ..., S\}$. To describe the cache placement, we define $I(b, s) \in \{0, 1\},\ \forall b \in \mathcal{B}^*, s \in \mathcal{S}$ as the indicator on whether content s is cached in server b. $I(b, s) = 1$ exists, if and only if content s is cached in base station b. Note that $I(0, s) = 1,\ \forall s \in \mathcal{S}$, as the remote server stores all the contents. For simplicity, we denote the cache size in each base station as K. We introduce $X_{ks}^b \in \{0, 1\}$ denoting whether base station b is the source server of requests for content s in base station k. The popularity of content s in base station b is defined as $p(b, s)$.

With the above definitions, we can formulate the problem as the following Integer Linear Program (ILP):

$$\min \sum_{b \in \mathcal{B}} \sum_{s \in \mathcal{S}} p(b, s) X_{ks}^b C(k, b) \tag{5}$$

$$s.t. \quad X_{ks}^b \leq I(b, s),\ \forall b, s, k; \tag{6}$$

$$I(0, s) = 1,\ \forall s; \tag{7}$$

$$\sum_{b=0}^{B} X_{ks}^b = 1,\ \forall k, s; \tag{8}$$

$$\sum_{s=1}^{S} I(b, s) \leq K,\ \forall b \in \mathcal{B}. \tag{9}$$

where constraint (6) indicates that users can retrieve a content from a base station only if it is cached, constraint (7) indicates that the remote server stores all the contents, constraint (8) indicates that requests initiated from one base station should be routed to one content source, and constraint (9) indicates that the cached content number should be no more than the cache size.

B. The Online Algorithm

In real-world scenarios, the user requests arrive one by one in an online fashion. The online algorithm has to decide which content to evict from the cache when a new content request is initiated. The decision has to be made when the new content arrives without the knowledge of the content popularity. Recall that we computed the cache priority in the DeepCache algorithm, which can be thought as the popularity of the corresponding content. In this way, we design an online cache algorithm based on DeepCache algorithm as follows.

A content request can be characterized by the initiated base station and the requested content. Formally, we denote $Req(b, s)$ as the content request initiated from base station b for content s. First, we introduce the content routing table. The content request will be routed to the source server with the lowest network delay. Formally, we define the source server for content request s in base station b^* as:

$$R(b^*, s) = \arg\min_{b} C(b^*, b),\ \forall b \in \{b | I(b, s) = 1\}. \tag{10}$$

The source servers are computed in the coordinator server, and the coordinator server maintains a routing table dynamically for each content in each base station.

When a new content request (not cached) in one base station arrives, we need to decide which content should be evicted from the cache based on both the content popularity and the variation of the network routing table. The content popularity

can be represented with the cache priority calculated by the DeepCache algorithm directly which is denoted as $p^{\dagger}(b, s)$, and we will introduce how to minimize the network delay via analyzing the variation of the routing table as follows.

When a new content request s_{new} arrives in base station b^*, the content replacement is triggered. The cache priority is computed in base station b^* via the DeepCache algorithm and transmitted to the coordinator server. Then, the coordinator server computes the delay reduction of caching s_{new} according to the content routing table as follows:

$$Cache(b^*, s_{new}) = \sum_{b' \in \mathcal{B}} [C(b', R(b', s_{new})) - C(b', b^*)]^+, \tag{11}$$

where $[x]^+ = \max(x, 0)$. We can observe that when the network delay between b' and b^* is lower than the delay between b' and $R(b', s_{new})$, the network delay can be reduced when caching s_{new} in base station b^*.

Similarly, we also need to compute the network delay of evicting a currently cached content s_c from base station b^*:

$$Evict(b^*, s_c) = \sum_{b' \in \mathcal{B}} [\min_{b \in \{b | I(b, s_c)=1\}/b^*} C(b', b) - C(b', b^*)]^+. \tag{12}$$

We can observe that when the base station b^* is the source of s_c in base station b', the network delay will increase if s_c is evicted from base station b^*.

In this way, the network delay reduction of replacing s_c with s_{new} in base station b^* can be computed as follows:

$$Replace(b^*, s_c, s_{new}) = Cache(b^*, s_{new}) p(b^*, s_{new}) - Evict(b^*, s_c) p(b^*, s_c) \tag{13}$$

If $Replace(b^*, s_c, s_{new}) \leq 0,\ \forall s_c \in \{s | I(b^*, s) = 1\}$, i.e., replacing arbitrary content with the new requested content will induce the network delay increasing, we do not replace the current cached content. Otherwise, we will evict the content with the maximum delay reduction, i.e.,

$$s_c = \arg\max_{s} Replace(b^*, s, s_{new}),\ s \in \{s | I(b^*, s) = 1\}. \tag{14}$$

Once the cache replacement happens, the routing table is updated according to equation (10).

C. Complexity Analysis

Recall that B denotes the number of base stations, and K denotes the cache size in each base station. $Cache(b^*, s_{new})$ is computed once and has a complexity of $O(B)$. When deciding which content to evict, $Replace(b, s', s)$ is executed K times, and the complexity of computing $Replace(b, s', s)$ is $O(B^2)$. Therefore, the overall complexity of executing the cooperative mechanism is $O(K + KB^2)$. In real-world scenarios, the value of K is usually small, while the content number S is large. Note that the complexity of the cooperation algorithm is independent of the content number, i.e., when the content number becomes larger, the complexity of the algorithm will not grow.

Algorithm 1: Online Cooperation Mechanism

Input: Network Delay Table $C(b_i, b_j)$, Content Routing Table $R(b, s)$, the content priority $p(b, s)$ computed by DeepCache, and the Content Request $Req(b, s)$.

Output: The cache replacement scheme.

```
MaxValue = 0
EvictContent = φ
1. Calculate Replacement Scheme:
Calculate Cache(b, s) based on equation (11).
for s' = {s|I(b, s) = 1} do
    Calculate Replace(b, s', s) based on equation (13).
    if Replace(b, s', s)>MaxValue then
        MaxValue = Replace(b, s', s)
        EvictContent = s'
2. Perform Replacement and Update Routing Table:
if EvictContent ≠ φ then
    Replace EvictContent with s.
    for s' = {s, EvictContent} do
        for b' = B do
            Update R(b', s') based on equation (10).
```

V. Experiment Results

We provide a mobile video request dataset to compare and evaluate algorithms with the state-of-the-art. We evaluate the performance of DeepCache on individual BS, and further evaluate the cooperative DeepCache algorithm in the edge network. We choose the following algorithms as baselines: LRU, LFU, LFUDA, Popcaching [9], and optimal [7].

A. The Mobile Video Request Dataset in Edge Network

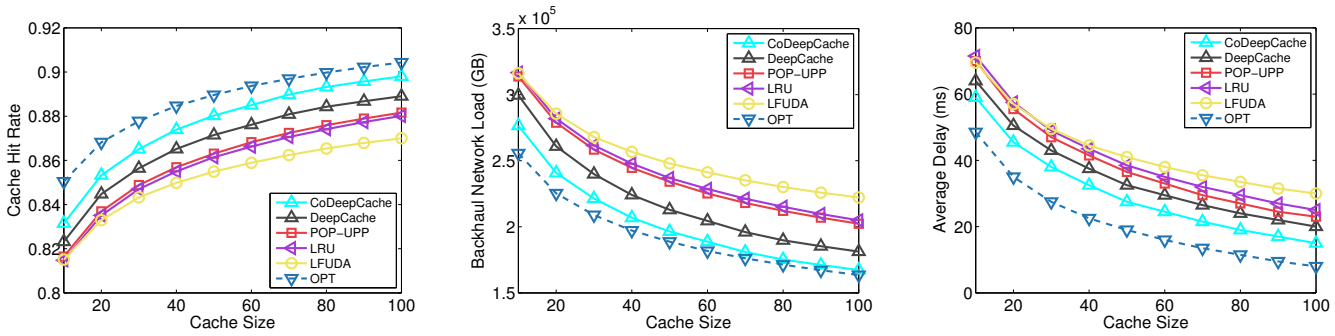
We collected the mobile video request dataset from iQiYi[2]. How users view videos in the mobile video streaming app has been recorded. The dataset spans 2 weeks and covers 2 million users watching 0.3 million unique videos in Beijing. In each trace item, the following information is recorded: (1) The device identifier, which is unique for different devices and can be used to track users; (2) The timestamp when the user starts to watch the video; (3) The location where the user watches the video collected from the device's built-in GPS function; (4) The title of the video, which is unique for different videos.

In order to investigate the content cache in the edge network, we jointly utilize the iQiYi dataset and the base station dataset. The base station dataset contains the locations and IDs of over 70 thousand base stations in Beijing. With the knowledge of the location and signal radius of a base station, we can map the user requests to the corresponding base station.

B. Parameter Settings

As the average video size in the dataset is about 1GB, we simply set the size of all videos as 1GB. Recall that the storage

[2]http://www.iqiyi.com/



(a) Cache hit rate with cache sizes. (b) Backhaul network traffic with cache sizes. (c) Average transmission delay with cache sizes.

Fig. 3: Performance comparison with different cache sizes in the cooperative edge network.

size of the MEC servers is quite varied in Table I, we discuss the cache size from 10GB to 100GB. Without additional mention, the default parameter settings in DeepCache are as follows: We set the default network layer as 2, and the cell number in each layer as 16. We further set the default input length as 5. With careful tuning, we set α as 0.1, and the default output length to calculate the label as 200.

C. Performance Comparison in the Edge Network

We conduct the performance comparison in the edge network. We denote the cooperative DeepCache algorithm as "CoDeepCache". We also introduce a cooperative mechanism as baseline based on User Preference Profiles (UPP). We employ the POP algorithm to derive the UPP, i.e., "POP-UPP". Without addition mention, the default base station number is set as 16, and the cache size is set as 10.

Fig. 3(a) shows that the CoDeepCache algorithm can achieve the best cache hit rate compared with baselines among all the cache sizes. The cache hit rate accounts for the local cache hit as well as the cache hit from other base stations. As the cache size gets larger, the cache hit rate is larger. We notice that under the cooperative edge network scenario, the cache hit rates of different algorithms are close. This happens as the base stations in the edge network can be seen as one node with large cache size. In this way, most of the user requests can be served by some base station in the edge network.

We further investigate the backhaul network load under different cache algorithms in Fig. 3(b). The backhaul network load is due to the requests served by the remote server. With the increase of the cache size, the backhaul network load can be reduced dramatically. The CoDeepCache algorithm can save 15%∼23% data traffic compared to POP-UPP, and save 12%∼16% data traffic compared to DeepCache. Then we investigate the average delay of a content request in Fig. 3(c). The CoDeepCache algorithm achieves the lowest transmission delay among all the cache sizes, and can reduce 14%∼ 22% transmission delay compared to the POP-UPP algorithm.

VI. Conclusion

We design a framework for cooperative edge caching in 5G. We propose a deep-learning-based caching algorithm DeepCache, which learns to make replacement decision by memorizing the former request sequence with the built-in memory cells. Then we design a cooperation mechanism for DeepCache to satisfy the cooperative edge caching scenario. Experiments on a large scale video dataset validates the effectiveness of our algorithms, which can reduce the transmission delay and the backhaul network load significantly.

Acknowledgement

The work of Haitian Pang and Lifeng Sun was supported by the National Natural Science Foundation of China (Grant No. 61472204 and 61521002), Beijing Key Laboratory of Networked Multimedia (Grant No. Z161100005016051), and Alibaba Cooperation Funding. The work of Jiangchuan Liu and Xiaoyi Fan was supported by an NSERC Engage Grant and an NSERC Discovery Grant.

References

[1] Cisco, "Cisco visual networking index: Global mobile data traffic forecast update 20160201 white paper," 2016.

[2] Q. Huang, K. Birman, R. van Renesse, W. Lloyd, S. Kumar, and H. C. Li, "An analysis of facebook photo caching," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 2013, pp. 167–181.

[3] S. Podlipnig and L. Böszörmenyi, "A survey of web cache replacement strategies," *ACM Computing Surveys (CSUR)*, vol. 35, no. 4, pp. 374–398, 2003.

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

[5] G. Ma, Z. Wang, M. Zhang, J. Ye, M. Chen, and W. Zhu, "Understanding performance of edge content caching for mobile video streaming," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 5, pp. 1076–1089, 2017.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[7] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, "Evaluation techniques for storage hierarchies," *IBM Systems journal*, vol. 9, no. 2, pp. 78–117, 1970.

[8] J. Jiang, R. Das, G. Ananthanarayanan, P. A. Chou, V. Padmanabhan, V. Sekar, E. Dominique, M. Goliszewski, D. Kukoleca, R. Vafin *et al.*, "Via: Improving internet telephony call quality using predictive relay selection," in *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*. ACM, 2016, pp. 286–299.

[9] S. Li, J. Xu, M. Van Der Schaar, and W. Li, "Popularity-driven content caching," in *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*. IEEE, 2016, pp. 1–9.