OA-Cache: Oracle Approximation based Cache Replacement at the Network Edge

Shuting Qiu, Qilin Fan, *Member, IEEE,* Xiuhua Li, *Member, IEEE,* Xu Zhang, *Member, IEEE,* Geyong Min, *Member, IEEE,* and Yongqiang Lyu, *Member, IEEE*

***Abstract*—With the explosive increase in mobile data traffic generated by various application services like video-on-demand and stringent quality of experience requirements of users, mobile edge caching is a promising paradigm to reduce delivery latency and network congestions by serving content requests locally. However, how to conduct cache replacement when the cache is full is a challenging issue when faced with enormous content volume and limited cache capacity at the network edge while the future request pattern is unknown ahead. In this paper, we propose a cache replacement algorithm based on the oracle approximation named OA-Cache in an end-to-end manner to maximize the cache hit rate. Specifically, we construct a complex model that uses a temporal convolutional network to capture the long and short dependencies between content requests. Then, an attention mechanism is adopted to find out the correlations between requests in the sliding window and cached contents. Instead of training a policy to mimic Belady that evicts the content with the longest reuse distance, we cast the learning task into a classification model to distinguish unpopular contents from popular ones. Finally, we apply the knowledge distillation approach to assist in transferring knowledge from a large pre-trained complex network to a lightweight network to readily accommodate to the network edge scenario. To validate the effectiveness of OA-Cache, we conduct extensive experiments on real-world datasets. The evaluation results demonstrate that OA-Cache can achieve better performance compared to candidate algorithms.**

***Index Terms*—Cache Replacement, Edge Caching, Imitation Learning, Knowledge Distillation.**

This work is supported in part by the National Key R & D Program of China (Grant No. 2022YFE0125400), the National NSFC (Grants No. 62102053 and 62072060), the Natural Science Foundation of Chongqing, China (Grant No. CSTB2022NSCQ-MSX1104), Key Research Program of Chongqing Science & Technology Commission (Grants No. cstc2021jscx-dxwtBX0019 and cstc2019jscx-zdztzxX0031), the EU Horizon 2020 research and innovation programme under the Marie Skodowska-Curie grant agreement No. 898588, and the Chongqing Key Laboratory of Digital Cinema Art Theory and Technology (Grant No. 2021KF01). This article reflects only the authors' view. The European Union Commission is not responsible for any use that may be made of the information it contains. *(Corresponding author: Qilin Fan.)*

S. Qiu and Q. Fan are with the School of Big Data and Software Engineering, Chongqing University, Chongqing 400044, China (email: qiushuting@cqu.edn.cn; fanqilin@cqu.edu.cn).

X. Li is with the School of Big Data and Software Engineering, Chongqing University, Chongqing, China 400000, and also with Haihe Laboratory of Information Technology Application Innovation, Tianjin, China 300072 (email: lixiuhua1988@gmail.com).

X. Zhang and G. Min are with the Department of Computer Science, University of Exeter, Exeter EX4 4QF, U.K. (e-mail: x.zhang6@exeter.ac.uk; g.min@exeter.ac.uk).

Y. Lyu is with the Beijing National Research Centre for Information Science and Technology (BNRist), Tsinghua University, Beijing 100084, China (e-mail: luyq@tsinghua.edu.cn).

I. Introduction

The Internet has witnessed the sky-rocketing growth of mobile data traffic with the popularization of various devices (e.g., portable cameras, smartphones) and application services (e.g., video-on-demand, virtual reality). According to a Cisco report [1], in 2022, the IP traffic flowing through the global network will exceed the total traffic of the entire 32 years from the first year of the Internet to the end of 2016. The enormous traffic brings tremendous pressure to the backbone network and significantly impacts users' quality of experience (QoE) [2], [3].

To this end, edge caching [4]–[6] emerges as a promising paradigm by caching popular contents at network edges, such as Wi-Fi APs and cellular base stations, and redirecting user requests to these local replicas instead of being served by the back-end/origin server. Thus, benefiting from caching, content requests can be served locally to reduce delivery latency and network congestions [7], [8]. However, compared with the enormous and increasing content volume, the cache size of the edge node is always restricted. It is impossible to cache all the contents locally. Hence, how to replace the cached content when the cache is full is a crucial issue.

However, unlike the traditional CDN caching [9], [10], edge caching has several characteristics [11], [12]: (i) limited resources: unlike cloud computing which has large capacity resources, the capacity of network edge is very limited. Therefore, the edge node highly relies on a well-designed caching strategy to cache "popular content" for good performance. (ii) dynamic request patterns: due to the mobility and personalized preferences of users, requests from edge networks show characteristics of fluctuations and bursts [13]. (iii) insufficient computational power: compared with large computing clusters such as cloud computing, the computing power of a single edge node is often limited, so it is not suitable for processing large-scale complex models. For these reasons, we need a lightweight cache replacement algorithm that can efficiently adapt to the characteristics of requests at the edge network.

The existing cache replacement algorithms are mostly based on heuristics such as recency-based [14], frequency-based [15], and so on. These methods are usually only applicable to a specific pattern and are not universal. Recently, caching algorithms based on learning have also been proposed, and many authors build deep models and use large training sets to predict content popularity or reuse distances to decide which content to be replaced or admitted [16], [17]. These approaches typically have high computational complexity and a long time for the online decision, thus unsuitable for highly dynamic edge networks.

In this paper, we propose a cache replacement algorithm based on the oracle approximation named OA-Cache in an end-to-end manner, without any explicit assumptions about the access pattern, and is well adapted to edge scenarios. Specifically, we first construct a complex model named OA-Cache-Teacher that fully learns the features of the accessed sequence, which uses a temporal convolution network to capture long and short dependencies of content requests. Then an attention mechanism is adopted to find out the correlations between requests in the sliding window and cached contents. Instead of training a policy to mimic Belady that evicts the content with the longest reuse distance, we cast the learning task into a classification model to distinguish unpopular contents from popular ones. Finally, we use knowledge distillation to extract a lightweight model with fewer parameters for quick decision-making named OA-Cache from OA-Cache-Teacher.

The contributions of this paper can be summarized as follows:

- We leverage a temporal convolution network (TCN) to characterize the multi-scale temporal features of content requests which can process the series in parallel and improve the computational efficiency compared to LSTM. An attention mechanism in a complex model is employed to dynamically learn the contextual relationships between each cached content and historical and current access.
- We propose an imitation learning architecture to calculate the eviction probability distribution to approximate the oracle policy. We innovatively introduce the binary cross-entropy loss function for training the policy to improve the prediction accuracy and versatility of the model.
- We apply the knowledge distillation approach to assist in transferring knowledge from a large pre-trained complex network to a lightweight network to accommodate to the network edge scenario.
- We conduct extensive experiments on real-world datasets from iQiYi, astar and MovieLens. The evaluation results demonstrate the better performance of OA-Cache compared to candidate algorithms.

The rest of this paper is organized as follows. Section II provides a brief overview of related work. Section III gives the system model and problem statement of cache replacement. In Section IV, we describe the methodology of OA-Cache. In Section V, we present the evaluation results. Finally, Section VI concludes the paper.

II. Related Work

Cache has attracted a lot of attention due to its advantages of reducing network latency and network backhaul overhead. The existing cache replacement algorithms can be classified into the following three categories.

A. *Heuristic-based algorithms.*

The most common cache replacement algorithm based on heuristics is least recent use (LRU) [14], which assumed that the recently accessed content will be used again soon, and some other policies focus on the frequency of accesses (i.e., LFU [15])) or the timing of cached content (i.e., FIFO [18]). S4LRU [19] divided the cache into four lists formed by LRU queues. ARC [20] adaptively divided the cache into two regions to separate pages that have been visited only once recently from frequent pages. LeCaR [21] recorded the recency and frequency of each cached data and utilized a regret minimization method to update the weights of these two policies. Zhou *et al.* [22] divided the cache into two parts and combined LFU and FIFO policies for partitioned caching. GDS [23] was a variant of LRU that takes content size and cost into account in a simple and non-parametric way for high performance.

These algorithms that follow heuristic rules are easy to implement. However, they may perform well in some access modes and poorly in others, failing to adapt to the dynamic content access modes of edge caches under real data flow.

B. *DRL-based algorithms.*

Some works advocated the use of deep reinforcement learning (DRL). For example, Zhong *et al.* [24] employed the deep deterministic policy gradient (DDPG) for training and the Wolpertinger architecture to deal with the large discrete action space. He *et al.* [25] designed a novel DRL algorithm for the studied QoE maximization problem and sought a balance between Q-values' accuracy and DRL acceleration's stability to improve the QoE satisfaction of intelligent caches. Wu *et al.* [26] modelled the problem as a Markov decision process and developed a new dynamic content update policy with the help of DRL to dynamically update the base station's cache. Zhou *et al.* [27] used DRL to learn the relationship between workload distribution and cache replacement policy distribution (including LRU and LFU). Ye *et al.* [28] designed a distributed bootstrap reinforcement learning framework to learn joint cache size scaling and replacement adaptation and used a distribution-guided regularization algorithm to maintain the intrinsic order of discrete variables. Sadeghi *et al.* [29] proposed a scalable DRL approach to learn Q-functions in an online manner to learn the optimal caching policy.

However, DRL-based algorithms require a tremendous number of learning samples and suffer from large-delayed rewards (cache hits), which can result in slow reaction times in highly dynamic environments and cannot adapt to scenarios where edge caching requires fast online decision-making.

C. *Supervised-based algorithms.*

Some other work adopts supervised learning that learns features in content requests, employs regression models, and performs the prediction task. For example, LRB [30] utilized a gradient boosting machine (GBM) to predict when the content will be next requested and incorporates a threshold to determine whether to evict the predicted content. DeepCache [12] used long short-term memory (LSTM) to predict content popularity, learning caching strategies from access sequences in real-time. Fedchenko *et al.* [31] employed feedforward neural networks (FNN) to predict content popularity and randomly re-evaluate the popularity of partially cached content. Belady

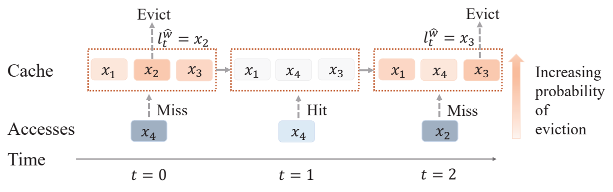


Fig. 1. Cache replacement. At $t = 0$, as access to content x_4 results in a miss, replacement policy π caches x_4 by evicting content x_2 with the highest probability from the contents $[x_1, x_2, x_3]$ in the cache. At $t = 1$, access to content x_4 is hit in the cache, so the cache state remains unchanged. At $t = 2$, access to content x_2 again results in a miss, so policy π caches x_2 by evicting content x_3 with the highest probability.

(oracle) algorithm [32] replaced the content with the largest reuse distance. However, as future information is unavailable in advance, it is a utopian optimal policy. Hawkeye [33] trained a binary classification model based on Belady to predict if the content is cache-friendly or cache-averse. But when all contents are cache-averse, it still uses LRU to determine which content to evict. PARROT [34] imitated Belady based on LSTM network architecture and utilized ranking loss and reuse distance loss for training. However, as the prediction task based on LSTM structure and ranking for all cached contents are computation-intensive and time-consuming, PARROT might be cumbersome for latency-sensitive applications.

Different from previous work, in this paper, we first propose a complex OA-Cache-Teacher model that leverages TCN with an attention mechanism to capture the features of content requests. Then we propose an imitation learning architecture and cast the task into a classification model to improve the versatility. Furthermore, we use a knowledge distillation approach to obtain a lightweight OA-Cache model from the OA-Cache-Teacher model.

III. System Model and Problem Statement

In this paper, let $C = \{c_1, \dots, c_m, \dots, c_M\}$ denote the set of M contents requested by the users. The sequence of requests for content is denoted as $X = [x_1, \dots, x_t, \dots, x_T]$, where x_t denotes the accessed content at moment t. We assume that all contents are unit-sized. The edge node can hold W contents. $L_t = [l_t^1, \dots, l_t^w, \dots, l_t^W]$ denotes the cache vector at time t. A binary vector $Z_t = [z_t^1, \dots, z_t^m, \dots, z_t^M]$ is an indicator of cached contents, where z_t^m represents whether content m is stored in the cache at time t. When a content request x_t arrives, the cache vector and indicator of cached content remain the same if it is hit in the cache, i.e., $L_{t+1} = L_t$, $Z_{t+1} = Z_t$. Otherwise, the cache evicts content $l_t^{\hat{w}}$ where $\hat{w}$ is selected by cache replacement policy π, and caches x_t. The transitions of cache vector and indicator of cached contents can be shown in Fig. 1 and modelled as follows:

$$l_{t+1}^w = \begin{cases} x_t, & w = \hat{w}, \\ l_t^w, & \text{otherwise.} \end{cases} \tag{1}$$

$$z_{t+1}^m = \begin{cases} 0, & m = l_t^{\hat{w}}, \\ 1, & m = x_t, \\ z_t^m, & \text{otherwise.} \end{cases} \tag{2}$$

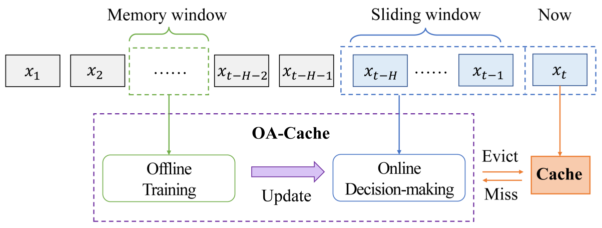


Fig. 2. Framework of OA-Cache.

We introduce HR(π) to represent the long-term average cache hit rate when T goes to infinity by adopting π, which is given by:

$$\mathrm{HR}(\pi) = \lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} z_t^{x_t}. \tag{3}$$

We aim to maximize HR(π) based on the constraint of cache capacity. Therefore, the cache replacement problem is formulated as:

$$\max_{\pi} \quad \mathrm{HR}(\pi) \tag{4}$$

$$s.t. \quad \sum_{m=1}^{M} z_t^m \le W, \quad t = 1, \dots, T, \tag{5}$$

$$z_t^m \in \{0, 1\}, \quad t = 1, \dots, T, \quad m = 1, \dots, M. \tag{6}$$

IV. Methodology

Van Roy *et al.* [35] proved that the Belady algorithm, which evicts the content with the longest reuse distance in the cache, is the optimal policy for the above problem. Therefore, the Belady algorithm is an upper bound for the performance of the cache replacement problem. However, it is infeasible as future information is unknown ahead in an online environment. To this end, in this paper, we propose a lightweight learning-based approach named OA-Cache to approximate Belady (oracle) algorithm at the network edge. The detailed design is presented below.

A. Framework of OA-Cache

As shown in Fig. 2, the framework of OA-Cache can be divided into two parts: offline training and online decision-making, which are elaborated as follows:

Offline training: If given the knowledge of future content requests, oracle could calculate the optimal cache eviction decision $\pi^*(w|L_t, x_t, x_{t+1}, \dots, x_T)$ when x_t arrives. In the offline training phase, experiences are sampled from the memory window. Our OA-Cache tries to learn a policy π^μ from experiences to approximate the optimal policy without using future accesses. OA-Cache updates the model parameters by calculating the loss function between the mimic policy and the expert policy composed of oracle and OA-Cache-Teacher. As new requests continually arrive, the memory window for training renews periodically.

Online decision-making: We use the trained OA-Cache model to make online cache replacement decisions. It is

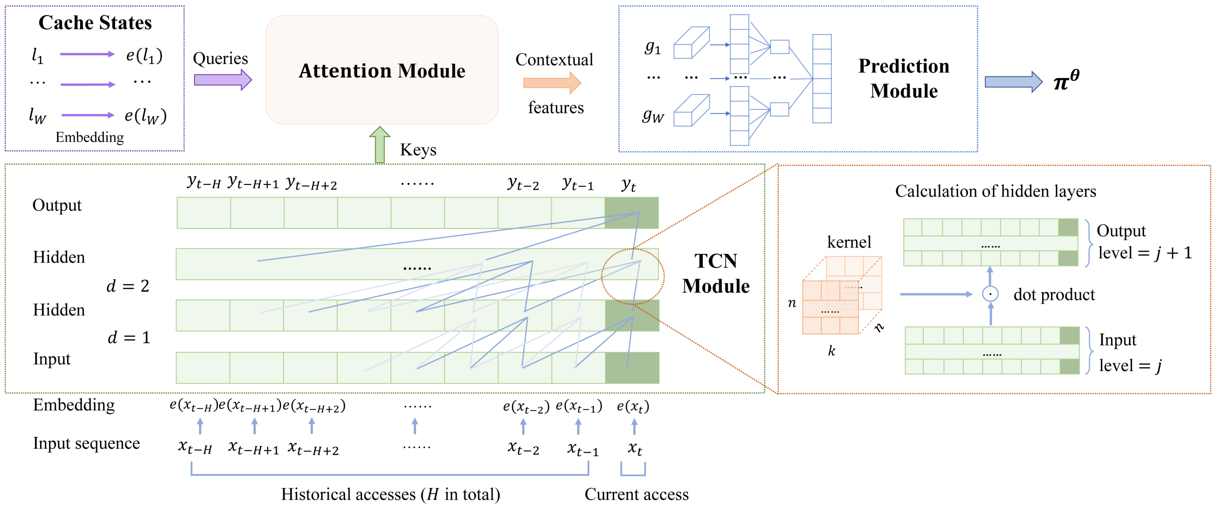


Fig. 3. OA-Cache-Teacher model.

impractical to perform the action based on all previous requests. To adapt to the latest request patterns, we apply a sliding window of length H to capture the temporal dependencies of requests. When x_t arrives while the cache declares a miss, OA-Cache feeds state s_t consisting of the current cache vector L_t, history accesses in the sliding window $[x_{t-H}, x_{t-H+1}, \ldots, x_{t-1}]$ and current access x_t into the deep neural network (DNN) and outputs the cache eviction decision $\hat{w} = \arg\max_w \pi^\mu(w|s_t)$. The cache replaces $l_t^{\hat{w}}$ with x_t. Then, with the arrival of the subsequent cache access x_{t+1} in the next time step, the sliding window slides one step further and contains $[x_{t-H+1}, x_{t-H+2}, ..., x_t]$ history accesses.

B. Model Architecture of OA-Cache-Teacher

The model architecture of OA-Cache-Teacher, illustrated in Fig. 3, consists of three components: (1) temporal convolutional network (TCN) module; (2) attention module and (3) prediction module. We will explain them in detail.

TCN module: We select the TCN [36] to capture the temporal dependencies of requests as it has the following advantages compared with recurrent neural networks (RNNs): i) Unlike the situations in RNN where the calculation for later time steps must wait for their predecessors to complete, convolutions in TCN can be calculated in parallel; ii) As filters are shared across the layers, TCN occupy less memory, especially for long sequences. OA-Cache-Teacher takes the embedding $E_t = [e(x_{t-H}), \ldots, e(x_{t-1}), e(x_t)] \in R^{n\times(H+1)}$ of the current access x_t and its H historical accesses $[x_{t-H}$, ..., $x_{t-1}]$ as the input to calculate the output of the last hidden layer of the TCN, where n is the content embedding dimension. Given the convolution filter $f : \{0, ..., k-1\} \to R^{n\times n}$, the dilation convolution operation F at the time t is defined as:

$$F(t) = \sum_{i=0}^{k-1} f(i) \cdot h^j_{t-d\cdot i}, \tag{7}$$

where k is the filter size, d is the dilation factor. We increase d exponentially with the depth of TCN (i.e., for j-th layer, $d = 2^j$). $h^j \in R^{n\times(H+1)}$ is the j-th hidden layer of TCN. $t - d\cdot i$ accounts for the direction of past information. Particularly, $h^0 = E_t$.

To keep the gradient from vanishing, a residual connection mechanism is utilized at each layer to ease the network training of TCN. Therefore, the j-th hidden layer h^j_t at the time t is calculated as:

$$h^j_t = \sigma(h^{j-1}_t + F(t)), \tag{8}$$

where σ is a nonlinearity activation function. The last hidden layer of TCN is the feature of the access sequences: $Y = [y_{t-H}, \ldots, y_t]$.

Attention module: We apply the scaled dot-product attention [37] in soft attention to calculate the contextual features of each cached content g_w, where each cached content embedding $e(l_w)$ is denoted as the query and hidden states Y as keys. Specifically, we compute the dot products of the query with all keys, divide each by $\sqrt{n}$, and apply a softmax function to obtain the weights a. Compared to other attention functions, the dot-product attention mechanism can be implemented with highly optimized matrix multiplication, thus showing the advantage of faster execution and higher spatial efficiency in practice. To eliminate the influence of dimensionality factors, we scale the results of dot-products for better access to sequence information. The specific calculation process of the context information is given as follows:

$$a_i = \text{softmax}\left(\frac{e(l_w)^T y_{t-H+i}}{\sqrt{n}}\right), \tag{9}$$

$$g_w = \sum_{i=0}^{H} a_i y_{t-H+i}. \tag{10}$$

Prediction module: The prediction module contains a flatten layer, a fully-connected layer, and a sigmoid layer. Specifically, the prediction module flattens each g_w to a row vector with dimension $R^{1\times n}$. Then the flattened contextual features of cached content are fed into a fully-connected layer with rectified linear unit (ReLU) activation function to obtain

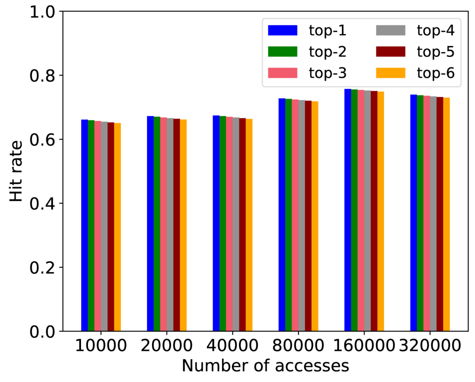


Fig. 4. Hit rate of oracle when evicting top-1 to top-6 content on iQiYi dataset.

the logits of each cached content. Finally, we calculate the eviction probability for each cached content $\pi^{\theta}(w|g_1,\dots,g_W)$ through a sigmoid layer. Therefore, we transform the prediction task into a 0-1 classification problem to distinguish popular contents from unpopular ones, regardless of the specific eviction probability of these contents.

C. Training Method of OA-Cache-Teacher

We sample consecutive access sequences of $[x_{t-H},\dots,x_{t+H}]$ from memory window and divide them into two discrete segments $X^{warm-up} = [x_{t-H},\dots,x_{t-1}]$ and $X^{validation} = [x_t,\dots,x_{t+H}]$. Combined with the cache status L_t, we train our cache replacement policy π^{θ} on the $X^{warm-up}$ and compute the loss $\mathcal{L}_{BCE}(\pi_i^{\theta},\pi_i^{*})$.

If the requested content misses, given the future access sequences, we could compute the reuse distance for each cached content and select the top-κ contents as the eviction candidates where $\kappa = W*\alpha$. Here, α denotes the percentage for classification. Fig. 4 illustrates how the hit rate varies on iQiYi dataset when evicting top-1 to top-6 content as the access sequences arrive with $W = 60$ and $\alpha = 10\%$. We can see that the differences are marginal among these distinct replacement policies. So evicting any one of the eviction candidates without paying attention to the relative priorities between these κ cached contents is feasible.

Compared to traditional imitation learning algorithms that learn only the optimal action of the oracle's policy, i.e., the cached content with the highest probability of eviction, this observation inspires us to use binary cross-entropy loss as our loss function in OA-Cache-Teacher to distinguish "cold" contents from "hot" contents to improve the prediction accuracy and versatility of the model. The loss $\mathcal{L}_{BCE}$ between OA-Cache-Teacher policy π_t^{θ} and oracle policy π_t^{*} is calculated as follows:

$$\mathcal{L}_{BCE}(\pi_t^{\theta},\pi_t^{*}) = -\frac{1}{W}\sum_{w=1}^{W}[\mathbb{1}_t(w)\log_2\pi_t^{\theta}(w) + (1-\mathbb{1}_t(w))\log_2(1-\pi_t^{\theta}(w)], \tag{11}$$

$$\mathbb{1}_t(w) = \begin{cases} 1, & w \in \text{top-}\kappa(\pi_t^{\theta}), \\ 0, & \text{otherwise}, \end{cases} \tag{12}$$

Algorithm 1: OA-Cache Algorithm

```
1  /**Offline**/
2  Pre-train the policy $\pi^{\theta}$ of OA-Cache-Teacher model;
3  Initialize the policy $\pi^{\mu}$ of OA-Cache model;
4  Set the number of epochs $R$; the number of sample times $S$; the length of historical accesses $H$;
5  for $epoch = 1,2,\dots,R$ do
6      for $s = 1,2,\dots,S$ do
7          Sample access sequences and divide them into two discreet segments from memory window;
8          Get the cache status $L_t$;
9          for $i = t, t+1,\dots,t+H$ do
10             Calculate the loss $\mathcal{L}_{\mathcal{KD}}(\pi^{\mu},\pi^{\theta},\pi^{*})$ according to Eq. (15);
11             $\kappa = W*\alpha$;
12             $candidates$ = top-$\kappa(\pi_i^{\mu})$;
13             Randomly evict $l_i^{\hat{w}}$ from $candidates$;
14             Update the cache status;
15             Update $\mu$ by loss $\mathcal{L}_{\mathcal{KD}}(\pi^{\mu},\pi^{\theta},\pi^{*})$;
16         end
17     end
18 end
19 /**Online**/
20 Obtain the policy $\pi^{\mu}$ from offline training;
21 Get current request $x_t$ and cache status $L_t$;
22 if $x_t$ not in $L_t$ then
23     if the cache is not full then
24         Caches $x_t$;
25     else
26         Feed $s_t$ that consists of $L_t$, historical accesses in sliding window $[x_{t-H},\dots,x_{t-1}]$ and $x_t$ into the DNN;
27         Compute eviction probability $\pi^{\mu}(w|s_t)$;
28         Evict $l_t^{\hat{w}}$ where $\hat{w} = \arg\max_w \pi(w|s_t)$ and cache $x_t$;
29     end
30 end
31 Update Cache status and move the sliding window one step forward;
```

where $\pi_t^{\theta}(w)$ is the probability for evicting cached content l_t^{w} computed by the OA-Cache-Teacher policy at time t. The function top-κ denotes the indices of top κ values of π_t^{θ}. $\mathbb{1}_t(w)$ denotes whether w-th cached content is labelled as the eviction candidates when given future information by oracle.

D. OA-Cache Algorithm at the Network Edge

In edge caching, we need a complex model to extract potential features from a large dataset while a lightweight model to make online decisions quickly. Therefore, there is some inconsistency between training and deployment. In order to reduce the number of model parameters and thus make the model ideally suited to the network edge scenario with guaranteed performance, we use a knowledge distillation [38] approach and combine it with a Belady strategy to transfer knowledge from a large pre-trained teacher network (OA-

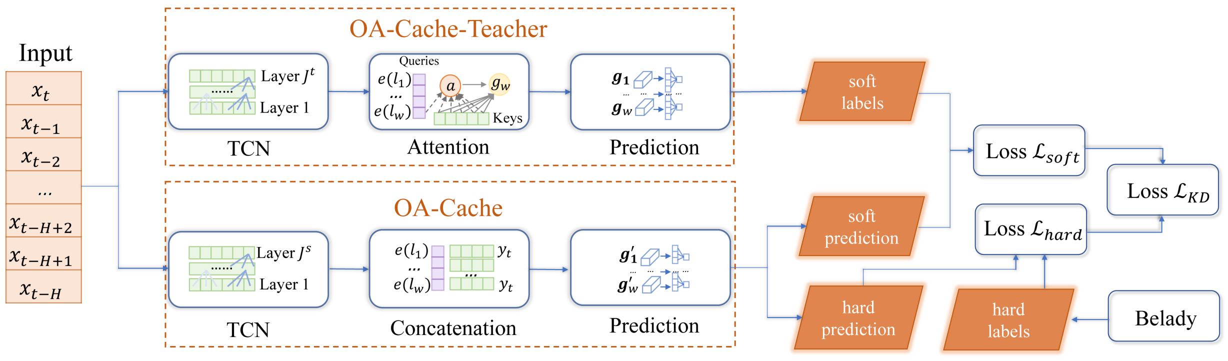


Fig. 5. The process of knowledge distillation.

Cache-Teacher) to a small student network (OA-Cache). The knowledge distillation process is shown in Fig. 5.

In this paper, the OA-Cache model is a small model with fewer parameters and a simple model structure. During the calculation process, we simply concatenate the current access sequence features y_t obtained by TCN with the embedding of each cached content, and then input the concatenated vector $g' = [g'_1, \dots, g'_W]$ into the fully connected layer to output the eviction probability π^μ of each cached content.

In the process of knowledge distillation, we use the outputs π^θ of the teacher network OA-Cache-Teacher as soft labels for supervising the training for the student network OA-Cache. This is mainly because, in addition to positive labels, negative labels also carry abundant potential information from the inductive inference of the OA-Cache-Teacher model. While in the traditional hard-label training method, all negative labels are treated equally. Then we use oracle-labeled hard labels π^* to assist the training of OA-Cache to effectively reduce the possibility of errors being propagated to the OA-Cache network when the OA-Cache-Teacher network sometimes makes incorrect decisions.

Furthermore, in our classification task, the sigmoid function is used to realize the conversion of logits to probabilities. The original sigmoid function is:

$$\pi_t^\mu(w) = \frac{1}{1 + exp(-z_t^w)}, \tag{13}$$

However, in the conversion process of the original sigmoid function, if the entropy of the probability distribution of the sigmoid output is relatively small, then the value of the negative label will be very close to 0, and the contribution to the loss function will be negligible. Therefore, we introduce the parameter "temperature" τ into the sigmoid function to amplify the information carried by the negative labels. The sigmoid function after adding the temperature variable is given by:

$$\pi_t^{\mu|\tau}(w) = \frac{1}{1 + exp(-z_t^w/\tau)}, \tag{14}$$

where z_t^w and $\pi_t^\mu(w)$ are the logits and eviction probabilities of the l_t^w cached content of the OA-Cache network at time t, respectively. τ determines the "softness of the teacher labels. It indicates that the higher the τ, the smoother the output probability distribution of the sigmoid, the greater the entropy of its distribution, and the higher the model's attention to negative labels.

The objective function of OA-Cache $\mathcal{L}_{KD}$ is calculated as follows:

$$\mathcal{L}_{KD} = \beta\mathcal{L}_{soft} + (1-\beta)\mathcal{L}_{hard}, \tag{15}$$

$$\mathcal{L}_{soft}(\pi_t^\mu, \pi_t^\theta) = -\frac{1}{W}\sum_{w=1}^{W}[\pi_t^{\theta|\tau}(w)\log\pi_t^{\mu|\tau}(w) + (1-\pi_t^{\theta|\tau}(w))\log(1-\pi_t^{\mu|\tau}(w)], \tag{16}$$

$$\mathcal{L}_{hard}(\pi_t^\mu, \pi_t^*) = -\frac{1}{W}\sum_{w=1}^{W}[\mathbb{1}_t(w)\log\pi_t^\mu(w) + (1-\mathbb{1}_t(w))\log(1-\pi_t^\mu(w)], \tag{17}$$

where $\pi_t^{\theta|\tau}(w) = \frac{1}{1+exp(-v_t^w/\tau)}$, v_t^w and $\pi_t^{\theta|\tau}(w)$ are the logits and eviction probabilities of the l_t^w cached content of the OA-Cache-Teacher network at time t, respectively. β is the ratio of the OA-Cache model affected by soft and hard labels. It is determined by the temperature τ, i.e., $\frac{\beta}{1-\beta} = \tau^2$, alleviating the problem of unbalanced gradient values of soft labels and hard labels due to the distillation temperature τ.

The offline and online cache replacement algorithms of OA-Cache are given in Algorithm 1. Let J^t and J^s denote the depth of TCN in OA-Cache-Teacher and OA-Cache, respectively. The time complexity of online decision-making of OA-Cache-Teacher is $O(J^tHkn^2 + WHn + Wn)$ and OA-Cache is $O(J^sHkn^2 + Wn)$, where $O(J^tHkn^2)$ and $O(J^sHkn^2)$ are the complexity of the TCN module, $O(WHn)$ is the complexity of the attention module, and $O(Wn)$ accounts for the prediction module.

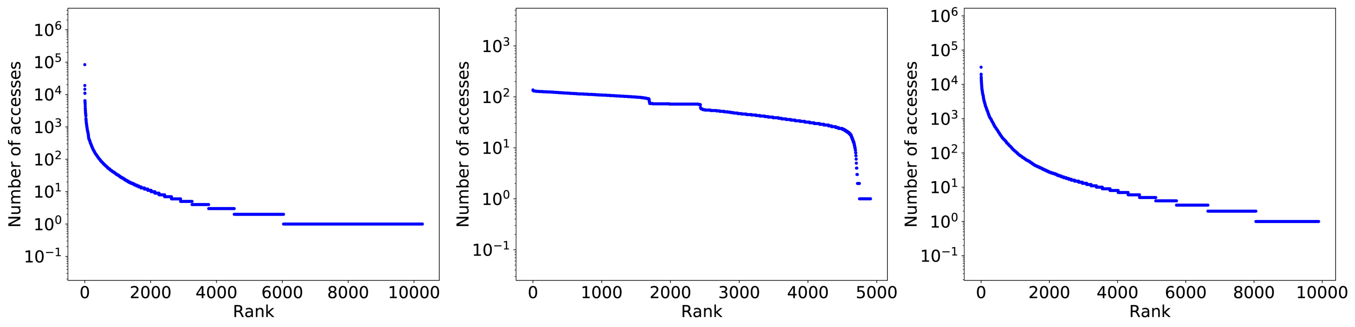
V. Evaluation

A. Dataset

We conduct the experiments on three real-world datasets, which are listed as follows:

iQiYi[1] contains 300,000 individual videos watched by 2 million users over two weeks which has been extensively used in previous work [12], [39]. Each entry contains the

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



(a) iQiYi (b) astar (c) MovieLens

Fig. 6. Number of accesses of the content versus its rank.

anonymized user information, the video content requested by each user, and the arrival time of the request.

astar[2] comes from the 2nd Cache Replacement Championship and contains memory access traces from the SPEC CPU2006 benchmark test. We subsample the traces by selecting 64 out of 2048 sets and filtering accesses to these sets.

MovieLens [40] contains 5-star rating and free-text tagging activities of 62,423 different movies by 162,541 different users from January 09, 1995 to November 21, 2019. We use the rating behaviors to simulate the content accesses and select corresponding requests of 10,000 movies from January 1, 2016. We split the dataset into an 80% training set and 20% test set.

In Fig. 6, contents are ranked in descending order by their number of accesses on iQiYi and astar, respectively. Each sample illustrates the number of accesses versus the rank of the corresponding content. We observe that the content popularity is highly skewed, following a Zipf-like distribution. However, content requests in iQiYi and astar show different access patterns. The figures illustrate that iQiYi contains more "overheated" contents while astar includes more "supercold" contents. Moreover, in Fig. 6(c) we can see that the data distribution of MovieLens is similar to that of iQiYi.

B. Experiment Settings

We randomly selected $M_1 = M3 = 10,000$ different content accesses from the iQiYi and MovieLens datasets and $M_2 = 5,000$ different content from the astar dataset. The cache percentage is the ratio of the cache size to the different contents selected on each dataset. We use a percentage of candidate evictions α_1 (10%) on iQiYi, α_2 (0.4%) on astar, and α_3 (0.3%) on MovieLens, learning rate (0.0001) and optimizer (Adam) to conduct the experiment. In the distillation process, we chose $\tau = 2$ for OA-Cache extraction. To calculate the network delay, we set a 10ms delay between the user and the edge node, and a 100ms delay between the edge node and the origin server [30]. When considering the backhaul traffic, we focus on calculating the backhaul cost between the edge node and the origin server. As all contents are assumed unit-sized, we set the backhaul traffic between the edge node and the origin server as 1GB for each missed content [12]. Unless explicitly stated, the experimental results are given with the above settings. OA-Cache is developed and tested on a PC equipped with a 14-core Intel(R) Core(TM) i7-12700H CPU@2.30 GHz, NVIDIA RTX 2050 graphics processor (4 GB of video memory), and 32 GB of RAM. Table I lists additional parametric information and descriptions.

[2]https://crc2.ece.tamu.edu/

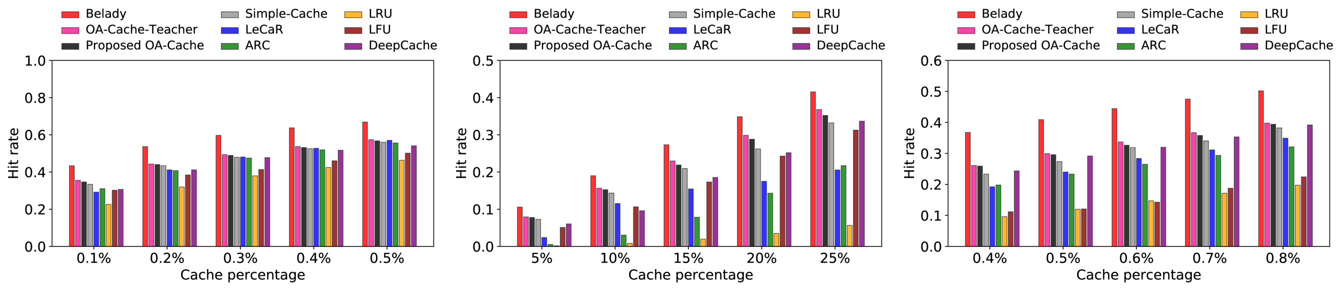
TABLE I
PARAMETER VALUES

Name	Value	Description
M_1	10000	Number of contents in iQiYi
M_2	5000	Number of contents in astar
M_3	10000	Number of contents in MovieLens
W_1/M_1	0.1%∼0.5%	Cache percentage for iQiYi
W_2/M_2	5%∼25%	Cache percentage for astar
W_3/M_3	0.4%∼0.8%	Cache percentage for MovieLens
n	3	Dimension of embedding and hiddden states
k	0.6	Kernel size
H	60	Length of historical accesses
α_1	1%∼30%	Percentage of candidate evictions for iQiYi
α_2	0.2%∼1.2%	Percentage of candidate evictions for astar
α_3	1%∼10%	Percentage of candidate evictions for MovieLens
J^t	3	Number of TCN layers in OA-Cache-Teacher
J^s	2	Number of TCN layers in OA-Cache
τ	2	Distillation temperature

C. Benchmarks and evaluation metrics

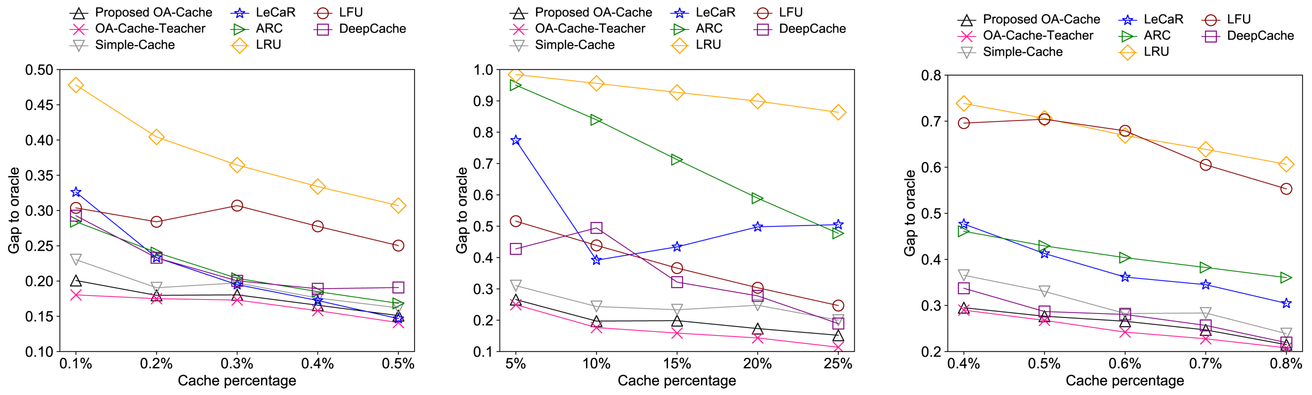
We compare our OA-Cache algorithm with the following algorithms:

- **Belady** [32]. It is an optimal offline algorithm that evicts the cached content with the largest reuse distance when the cache storage is full and the requested content misses.
- **OA-Cache-Teacher**. It is the complex model proposed for knowledge distillation in this paper, which has a higher number of parameters compared to OA-Cache.
- **Simple-Cache**. It has the same network structure as OA-Cache, but only uses hard labels marked by Belady for training instead of soft labels output by OA-Cache-Teacher.
- **DeepCache** [12]. It utilizes the memory unit of LSTM to characterize the temporal dependencies among requests and learns the priority of each content.



(a) Hit rate on iQiYi (b) Hit rate on astar (c) Hit rate on MovieLens

Fig. 7. Cache hit rate under varying cache percentages on different datasets.



(a) GtO on iQiYi (b) GtO on astar (c) GtO on MovieLens

Fig. 8. GtO under varying cache percentages on different datasets.

- **LeCaR** [21]. It utilizes two basic eviction policies (i.e., LRU and LFU). The weights of these two policies are updated using the regret minimization technique, and the policy with the higher weight is used each time for cache eviction decisions.
- **ARC** [20]. It splits the cache into two parts. One catches the contents that have been accessed only once and the other caches the contents that have been accessed many times. ARC also record LRU-based eviction history of these two parts through which the recency and frequency priorities are adjusted and the size of these two parts are changed dynamically.
- **LRU** [14]. It evicts the cached content which has been least requested recently when the cache storage is full and the requested content misses.
- **LFU** [15]. It evicts the cached content which has been least frequently requested when the cache storage is full and the requested content misses.

Our evaluation is based on the following metrics:

- Edge hit rate. This metric is calculated as the total number of edge hit requests divided by the total number of requests. It reflects the ratio of content requests served by the edge node instead of the origin server.
- Gap to oracle (GtO). It measures how well the algorithms approximate the Belady strategy, which is given by:

$$\mathrm{GtO} = \frac{r_{\mathrm{Belady}} - r_{\mathrm{Alg}}}{r_{\mathrm{Belady}}}, \tag{18}$$

where r_{Belady} denotes the hit rate of Belady algorithm and r_{Alg} denotes the hit rate of a specific algorithm.

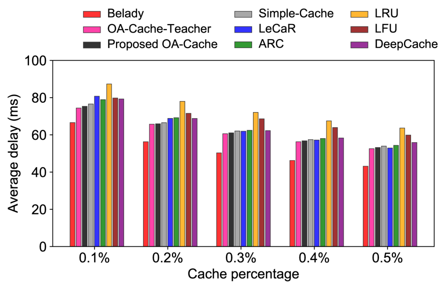
- Average access delay. It is calculated as the average waiting time required to retrieve the content for all requests from the edge node or the origin server.
- Average backhaul traffic. This metric calculates the average total traffic cost of all requests between the edge node and the origin server.

In addition, to demonstrate the effectiveness of our distillation method, we also evaluate some algorithms in terms of floating-point computation, parameters amount and average decision time.

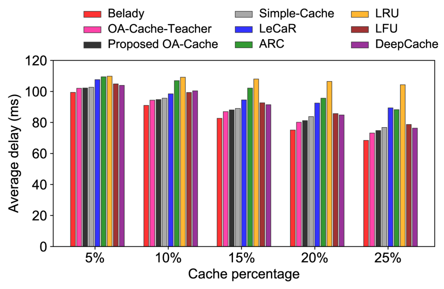
D. Performance Comparison

Edge hit rate and Gto. Fig. 7 and Fig. 8 show the hit rate and GtO performance under varying cache percentages on the iQiYi, astar and MovieLens datasets, respectively. Here we mainly analyze the GtO metric because the hit rate and GtO are interconvertible. As can be seen from the figures, the performance gap between OA-Cache-Teacher and Belady is less than that of the other candidate algorithms in all cases, and OA-Cache can achieve an effect close to OA-Cache-Teacher.

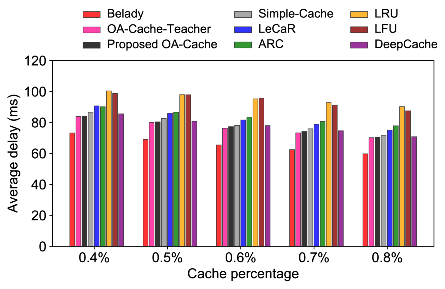
Specifically, for iQiYi dataset shown in Fig. 8(a), when the cache percentage is 0.1%, the GtO of OA-Cache-Teacher and OA-Cache is only 18% and 20%, while the GtO of Simple-Cache, ARC, DeepCache, LFU, LeCaR and LRU is 28%, 29%,



(a) Access delay on iQiYi

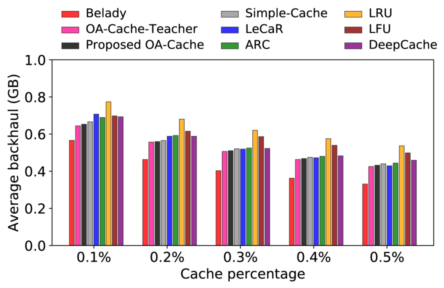


(b) Access delay on astar

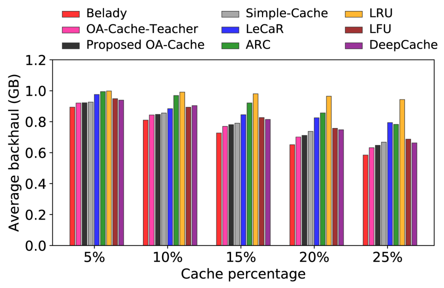


(c) Access delay on MovieLens

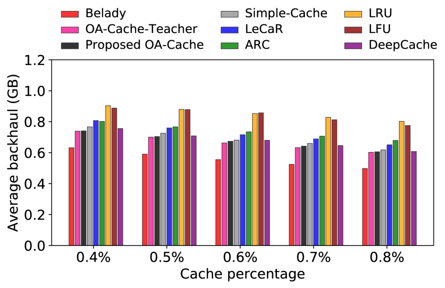
Fig. 9. Average access delay under varying cache percentages on different datasets.



(a) Backhaul traffic on iQiYi



(b) Backhaul traffic on astar



(c) Backhaul traffic on MovieLens

Fig. 10. Average backhaul traffic under varying cache percentages on different datasets.

30%, 32%, and 47%, respectively. When the cache percentage is 0.2%, the performance of OA-Cache is closer to that of OA-Cache-Teacher. The gap between the two and Belady is 17.4% and 17.9%, respectively, while the GtO of Simple-Cache, DeepCache, LeCaR, ARC, LFU, and LRU is 19%, 23.2%, 23.3%, 24%, 28%, and 40%.

For astar dataset shown in Fig. 8(b), we can observe that at 5% cache percentage, the GtO of OA-Cache-Teacher and OA-Cache is only 24.9% and 26.5%. In comparison, the GtO of Simple-Cache, DeepCache, LFU and LeCaR is 31%, 42.7%, 51.6% and 77%, respectively, while the GtO of other algorithms is more than 95%. Even when the cache percentage is large (i.e., 25%), the GtO of OA-Cache-Teacher and OA-Cache is much lower than that of other algorithms.

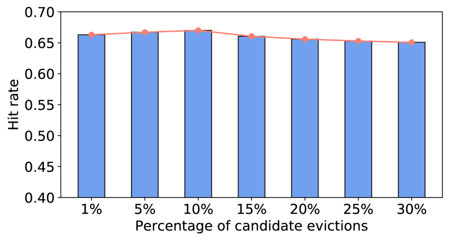
For MovieLens dataset, Fig. 8(c) illustrates that the GtO of OA-Cache-Teacher and OA-Cache at 0.4% cache percentage is only 28.9% and 29.5%, while the GtO of DeepCache, Simple-Cache, ARC, LeCaR, LFU and LRU is 33.7%, 36.5%, 46.1%, 47.6%, 69.5%, and 73.8%, respectively. As the cache percentage increases, OA-Cache can always maintain a low GtO value.

It can be seen that our proposed complex algorithm OA-Cache-Teacher can make decisions that best approximate Belady's strategy due to the richness of its parameters and structure. The algorithm OA-Cache extracted by our knowledge distillation method can also achieve good results. Compared to the Simple-Cache network trained with only traditional hard labels, OA-Cache more closely approximates the oracle's policy. Therefore, knowledge distillation method is superior in helping the lightweight model to learn the knowledge of experts efficiently.

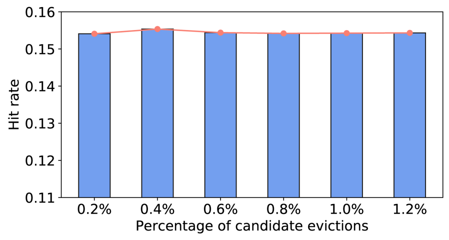
Average access delay. The cache hit rate of edge nodes directly affects the average delay of retrieving the accessed content. When the requested content is missed in the cache, the edge node queries the origin server for the corresponding content, resulting in increased delay. Therefore, when the cache hit rate is higher, the corresponding network delay is lower. In Fig. 9, we compare the average delay for algorithms under different cache percentages on three datasets. In the iQiYi dataset, the delay of OA-Cache is reduced by 0.8%∼1.7% compared to Simple-Cache; compared to ARC and LRU, it can reduce the delay by 2.1%∼4.9% and 15.9%∼19.6%, respectively. In the MovieLens dataset, compared to DeepCache, Simple-Cache and LRU, the delay of OA-Cache is reduced by 0.3%∼1.8%, 0.9%∼3% and 19%∼27%, respectively. In the astar dataset, when the cache percentage is 5%, the delay of OA-Cache is reduced by 0.45% compared to Simple-Cache and more than 1.6% compared to other algorithms. It can be seen that OA-Cache is able to store popular contents and reduce access delay.

Average backhaul traffic. When the cache hit rate increases, the average backhaul traffic for retrieving specific content decreases. Fig. 10 shows the average backhaul traffic comparison of algorithms under varying cache percentages on the three datasets. It can be seen that our proposed OA-Cache has lower backhaul traffic compared to the rest of the candidate algorithms, second only to Belady and OA-Cache-Teacher. In the iQiYi dataset, the backhaul traffic of OA-Cache is reduced by more than 1% compared to Simple-Cache, and by more than 18% compared to LRU. In the astar and MovieLens datasets, OA-Cache also yields lower backhaul traffic.

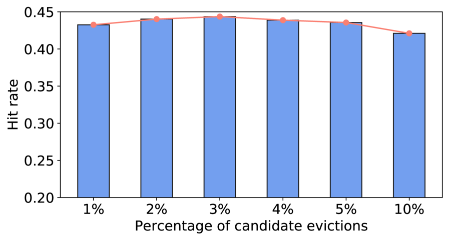
Quantitative analysis. To demonstrate that our OA-Cache



(a) Variable α_1 on iQiYi

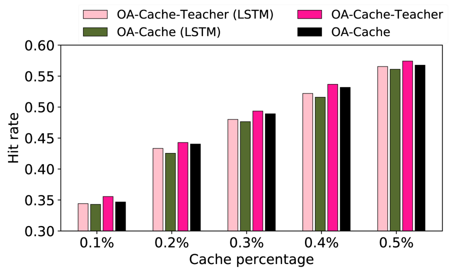


(b) Variable α_2 on astar

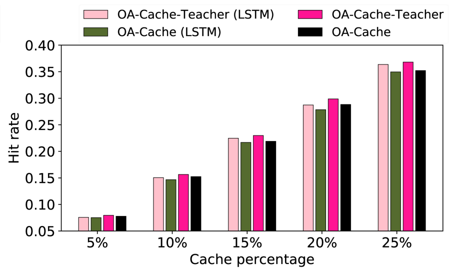


(c) Variable α_3 on MovieLens

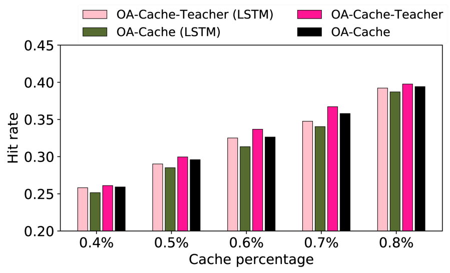
Fig. 11. The effect of the variable α on the hit rate.



(a) On iQiYi



(b) On astar



(c) On MovieLens

Fig. 12. Comparison of cache hit rate after ablation using LSTM instead of TCN.

TABLE II
QUANTITATIVE ANALYSIS

Algorithm	#Flops (M)	#Params (K)	Average Decision Time (ms)
DeepCache	0.5943	644.165	48.5075
OA-Cache-Teacher	12.6692	554.881	11.4813
OA-Cache-Teacher (LSTM)	4.1434	571.777	14.2110
OA-Cache (LSTM)	0.0274	63.285	5.0945
Proposed OA-Cache	**0.0373**	**64.121**	**1.1653**

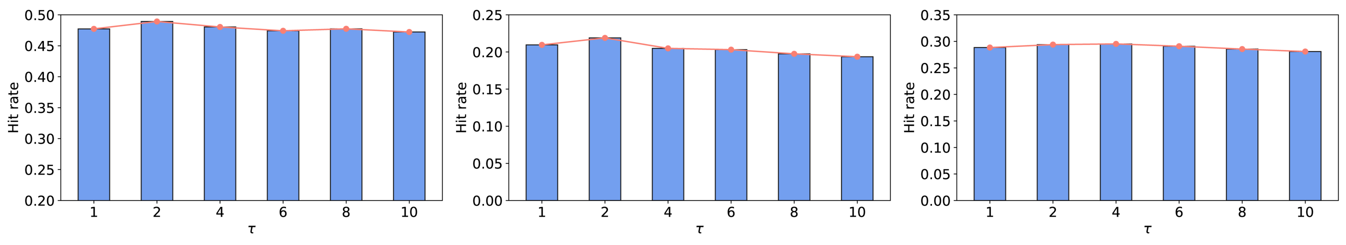
model is suitable for edge scenarios, we conduct quantitative analysis from three perspectives: floating-point computation amount, parameter amount, and average decision time. The specific values are given in Table II. As can be seen from Table II, compared with the complex model OA-Cache-Teacher and the DeepCache algorithm based on LSTM, our proposed OA-Cache model has less floating-point computation, shorter average decision-making time, and can quickly make replacement decisions, which is suitable for edge networks with highly dynamic requests. Also, our model has fewer parameters, which can meet the requirements of lightweight models for edge networks. Therefore, even though the OA-Cache, extracted using the knowledge distillation approach, sacrifices a small hit rate compared to the complex model OA-Cache-Teacher, it is more suitable for edge networks than OA-Cache-Teacher and can also make appropriate decisions.

To sum up, OA-Cache algorithm achieves good results regardless of whether the dataset contains "overheated" or "supercold" content, especially on smaller cache percentages. This is mainly because algorithms (i.e., LFU, LRU or their variants) will take lagged time to evict the cached unpopular contents. So caching these "cold" contents is quite likely to generate cache misses for caches with small capacity. DeepCache learns the caching priority of all the contents. However, the task of learning to rank for the dynamic request pattern might yield inconsistency when applied in the online scenario. In contrast, OA-Cache learns the context of each cached content based on historical information and simultaneously learns knowledge from OA-Cache-Teacher and Belady algorithms to evict the "cold" content in the cache with high probability and carries out the robust classification task for training thus rendering a higher hit rate and reducing the delay, backhaul traffic, and decision time.

E. Ablations Study

To better explain the relative importance of the components of our proposed approach, we design two ablation experiments on three datasets. One is to verify that randomly evicting top-κ content during the training phase is superior to evicting the furthest one. The other is to prove that the TCN is more efficient than the LSTM.

Component of evicting among top-κ. As shown in Fig. 11, we measure the effect of variables α_1, α_2 and α_3 on the hit rate of OA-Cache under cache percentage $W_1/M_1 = 1\%$, $W_2/M_2 = 10\%$ and $W_3/M_3 = 1\%$ on iQiYi, astar and MovieLens datasets, respectively. In iQiYi, OA-Cache can always maintain a relatively high hit rate when α_1 varies from 1% to 30%. OA-Cache achieves the highest hit rate value when $\alpha_1 = 10\%$, better than evicting the cached content with the highest eviction probability ($\alpha_1 = 1\%$) directly. The results and conclusions are similar for MovieLens dataset. It indicates that compared to the oracle policy that evicts the top-1 cached content, OA-Cache could improve the performance via selecting a random one from the candidate



(a) Variable τ in iQiYi (b) Variable τ in astar (c) Variable τ in MovieLens

Fig. 13. The effect of the variable τ on the hit rate.

set for eviction. However, the hit rate gradually decreases as α_1 and α_3 increase. In astar, OA-Cache also achieves better results on α_2 = 0.4% and shows very similar performance on the remaining percentages. It might be because the flattening popularity distribution in astar dataset enables the parameter of partition percentage in OA-Cache insensitive. It can be seen that our approach using random eviction from a candidate set for model training has certain advantages in cache replacement problem.

Component of TCN. Fig. 12 summarizes our results when using LSTM instead of TCN for ablation on three datasets. At $W1/M1 = 0.3\%$, $W2/M2 = 15\%$ and $W3/M3 = 0.6\%$, for the astar, iQiYi and Movielens datasets, compared with OA-Cache (LSTM) using LSTM, the hit rate of OA-Cache using TCN to obtain access features is improved by 1%, 2.6% and 3.9%, respectively. In addition, we analyze the performance in running time of the model after ablation using LSTM. As can be seen in Table II, OA-Cache (LSTM) is slightly smaller than OA-Cache in terms of floating point count and number of parameters, but has a longer average decision time. This is mainly due to the efficient parallel computing capability during TCN convolution. Therefore, the TCN component is more computationally efficient than LSTM, whose temporal dependencies are step-wise.

F. Effect of Parameters

In this subsection, we perform the analysis of the effect of different temperatures during distillation and sliding window sizes for OA-Cache on the iQiYi, astar, and MovieLens datasets using cache percentages $W1/M1 = 0.3\%$, $W2/M2 = 15\%$ and $W3/M3 = 0.5\%$, respectively.

Different distillation temperatures. From Fig. 13, we can see that when $\tau = 2$, OA-Cache achieves the best results on both iQiYi and astar datasets, and achieves similar results on MovieLens dataset as $\tau = 4$. As τ increases, the overall hit rate shows a downward trend. This is mainly because when the distillation temperature is too high, the probability distribution of soft labels will be too smooth and the available training information might be lost. For this reason, within a reasonable range of τ, we choose τ = 2 as our distillation temperature.

Different sliding window sizes. As shown in Fig. 14, we can find that OA-Cache achieves good results on all three different datasets when the sliding window size is greater than 60. In particular, when the sliding window size varies from 60 to 100, the improvement in the hit rate metric for the OA-Cache is relatively small. Therefore, considering that a larger sliding window size requires more storage and computing resources, we decide to use a sliding window size of $H = 60$ in our model. In addition, it can be observed that when H is small (i.e., $H = 20$), OA-Cache has a lower hit rate as it cannot fully utilize the historical access information. Meanwhile, when H is larger (i.e., $H = 120$), OA-Cache may pay less attention to the recently accessed data due to excessive attention to historical information, which might also lead to a lower hit rate for astar and MovieLens datasets.

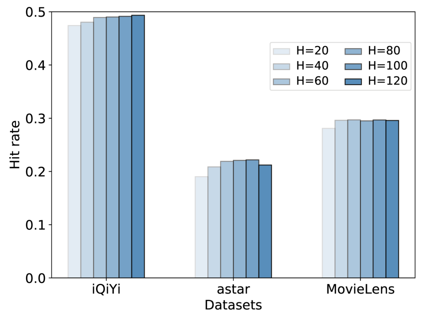


Fig. 14. The effect of variable H for the OA-Cache on the hit rate metric.

VI. Conclusion

In this paper, we have proposed a novel cache replacement algorithm named OA-Cache to approximate the oracle policy. By leveraging TCN with an attention mechanism, we have firstly designed a complex OA-Cache-Teacher model that learns the contextual relationships between cached contents and accesses with multi-scale temporal features. Then, we have cast the imitation learning task into a classification model which utilizes the binary cross-entropy as a loss function for training. Furthermore, we have applied the knowledge distillation approach to extract a lightweight model from a large pre-trained OA-Cache-Teacher to accommodate to the network edge scenario. The extensive experiments on real-world datasets have verified the effectiveness and robustness of OA-Cache. In our future work, we plan to consider the joint problem of cache admission and cache replacement to promote the hit rate. Designing the learning-based approach for mixed cooperative-competitive caching of multiple caches is also our future research focus.

References

[1] T. Barnett, S. Jain, U. Andra, and T. Khurana, "Cisco visual networking index (VNI) complete forecast update, 2017–2022," *Americas/EMEAR Cisco Knowledge Network (CKN) Presentation*, pp. 1–30, 2018.

[2] X. Ma, Q. Li, Y. Jiang, G.-M. Muntean, and L. Zou, "Learning-based joint QoE optimization for adaptive video streaming based on smart edge," *IEEE Transactions on Network and Service Management*, vol. 19, no. 2, pp. 1789–1806, Jan. 2022.

[3] H. Wang, K. Wu, J. Wang, and G. Tang, "Rldish: Edge-assisted QoE optimization of HTTP live streaming with reinforcement learning," in *Proc. of IEEE INFOCOM*, Aug. 2020, pp. 706–715.

[4] A. Lekharu, M. Jain, A. Sur, and A. Sarkar, "Deep learning model for content aware caching at MEC servers," *IEEE Transactions on Network and Service Management*, vol. 19, no. 2, pp. 1413–1425, Dec. 2021.

[5] N. Zhang, W. Wang, P. Zhou, and A. Huang, "Delay-optimal edge caching with imperfect content fetching via stochastic learning," *IEEE Transactions on Network and Service Management*, vol. 19, no. 1, pp. 338–352, Oct. 2021.

[6] S. Bayhan, S. Maghsudi, and A. Zubow, "Edgedash: Exploiting network-assisted adaptive video streaming for edge caching," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1732–1745, Nov. 2020.

[7] R. Li, K. Matsuzono, H. Asaeda, and X. Fu, "Achieving high throughput for heterogeneous networks with consecutive caching and adaptive retrieval," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 4, pp. 2443–2455, Jul. 2020.

[8] R. Immich, L. Villas, L. Bittencourt, and E. Madeira, "Multi-tier edge-to-cloud architecture for adaptive video delivery," in *Proc. of FiCloud*, Aug. 2019, pp. 23–30.

[9] S. T. Thomdapu, P. Katiyar, and K. Rajawat, "Dynamic cache management in content delivery networks," *Computer Networks*, vol. 187, p. 107822, Mar. 2021.

[10] Y. Sun, Z. Guo, S. Dou, and Y. Xia, "Video quality and popularity-aware video caching in content delivery networks," in *Proc. of IEEE ICWS*, Sep. 2021, pp. 648–650.

[11] Q. Fan, X. Li, J. Li, Q. He, K. Wang, and J. Wen, "PA-Cache: Evolving learning-based popularity-aware content caching in edge networks," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1746–1757, Jan. 2021.

[12] H. Pang, J. Liu, X. Fan, and L. Sun, "Toward smart and cooperative edge caching for 5G networks: A deep learning based approach," in *Proc. of IWQoS*, Jun. 2018, pp. 1–6.

[13] Y. Guan, X. Zhang, and Z. Guo, "CACA: learning-based content-aware cache admission for video content in edge caching," in *Proc. of ACM MM*, Oct. 2019, pp. 456–464.

[14] M. Ahmed, S. Traverso, P. Giaccone, E. Leonardi, and S. Niccolini, "Analyzing the performance of LRU caches under non-stationary traffic patterns," *arXiv preprint arXiv:1301.4909*, Jan. 2013.

[15] A. Jaleel, K. B. Theobald, S. C. Steely Jr, and J. Emer, "High performance cache replacement using re-reference interval prediction (RRIP)," *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3, pp. 60–71, Jun. 2010.

[16] G. Yan and J. Li, "RL-bélády: A unified learning framework for content caching," in *Proc. of ACM MM*, Oct. 2020, pp. 1009–1017.

[17] V. Kirilin, A. Sundarrajan, S. Gorinsky, and R. K. Sitaraman, "RL-cache: Learning-based cache admission for content delivery," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 10, pp. 2372–2385, Jun. 2020.

[18] K. Kogan, A. López-Ortiz, S. I. Nikolenko, and A. V. Sirotkin, "Online scheduling FIFO policies with admission and push-out," *Theory of Computing Systems*, vol. 58, no. 2, pp. 322–344, Apr. 2016.

[19] Q. Huang, K. Birman, R. Van Renesse, W. Lloyd, S. Kumar, and H. C. Li, "An analysis of facebook photo caching," in *Proc. of ACM SOSP*, Nov. 2013, pp. 167–181.

[20] N. Megiddo and D. S. Modha, "ARC: A self-tuning, low overhead replacement cache," in *Proc. of FAST*, Mar. 2003, pp. 1–6.

[21] G. Vietri, L. V. Rodriguez, W. A. Martinez, S. Lyons, J. Liu, R. Rangaswami, M. Zhao, and G. Narasimhan, "Driving cache replacement with ML-based LeCaR," in *Proc. of HotStorage*, Jul. 2018, pp. 1–6.

[22] Y. Zhou, L. Chen, C. Yang, and D. M. Chiu, "Video popularity dynamics and its implication for replication," *IEEE Transactions on Multimedia*, vol. 17, no. 8, pp. 1273–1285, Aug. 2015.

[23] P. Cao and S. Irani, "Cost-aware www proxy caching algorithms," in *Proc. of USITS*, Dec. 1997.

[24] C. Zhong, M. C. Gursoy, and S. Velipasalar, "A deep reinforcement learning-based framework for content caching," in *Proc. of CISS*, Mar. 2018, pp. 1–6.

[25] X. He, K. Wang, and W. Xu, "QoE-driven content-centric caching with deep reinforcement learning in edge-enabled IoT," *IEEE Computational Intelligence Magazine*, vol. 14, no. 4, pp. 12–20, Nov. 2019.

[26] P. Wu, J. Li, L. Shi, M. Ding, K. Cai, and F. Yang, "Dynamic content update for wireless edge caching via deep reinforcement learning," *IEEE Communications Letters*, vol. 23, no. 10, pp. 1773–1777, Jul. 2019.

[27] Y. Zhou, F. Wang, Z. Shi, and D. Feng, "An end-to-end automatic cache replacement policy using deep reinforcement learning," in *Proc. of ICAPS*, vol. 32, Jun. 2022, pp. 537–545.

[28] J. Ye, Z. Li, Z. Wang, Z. Zheng, H. Hu, and W. Zhu, "Joint cache size scaling and replacement adaptation for small content providers," in *Proc. of IEEE INFOCOM*, May. 2021, pp. 1–10.

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

[30] Z. Song, D. S. Berger, K. Li, A. Shaikh, W. Lloyd, S. Ghorbani, C. Kim, A. Akella, A. Krishnamurthy, E. Witchel *et al.*, "Learning relaxed Belady for content distribution network caching," in *Proc. of NSDI*, Feb. 2020, pp. 529–544.

[31] V. Fedchenko, G. Neglia, and B. Ribeiro, "Feedforward neural networks for caching: enough or too much?" *ACM SIGMETRICS Performance Evaluation Review*, vol. 46, no. 3, pp. 139–142, Dec. 2019.

[32] L. A. Belady, "A study of replacement algorithms for a virtual-storage computer," *IBM Systems Journal*, vol. 5, no. 2, pp. 78–101, 1966.

[33] A. Jain and C. Lin, "Back to the future: leveraging Belady's algorithm for improved cache replacement," in *Proc. of ISCA*, Aug. 2016, pp. 78–89.

[34] E. Liu, M. Hashemi, K. Swersky, P. Ranganathan, and J. Ahn, "An imitation learning approach for cache replacement," in *Proc. of ICML*, Jul. 2020, pp. 6237–6247.

[35] B. Van Roy, "A short proof of optimality for the min cache replacement algorithm," *Information processing letters*, vol. 102, no. 2-3, pp. 72–73, Apr. 2007.

[36] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *arXiv preprint arXiv:1803.01271*, Apr. 2018.

[37] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, Jun. 2017.

[38] G. Hinton, O. Vinyals, J. Dean *et al.*, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, vol. 2, no. 7, Mar. 2015.

[39] 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, Mar. 2017.

[40] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, dec 2015.

Shuting Qiu is currently working towards her B.E. degree from the School of Big Data and Software Engineering, Chongqing University, Chongqing, China. Her research interests include network optimization, information-centric networking, software defined networking and edge computing.

Qilin Fan (Member, IEEE) is currently a Associate Professor in the School of Big Data and Software Engineering, Chongqing University, Chongqing, China. She received the B.E. degree in the College of Software Engineering, Sichuan University, Chengdu, China, in 2011, and the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 2017. Her research interests include network optimization, mobile edge computing and caching, network virtualization and machine learning.

Xiuhua Li (Member, IEEE) received the B.S. degree from the Honors School, Harbin Institute of Technology, Harbin, China, in 2011, the M.S. degree from the School of Electronics and Information Engineering, Harbin Institute of Technology, in 2013, and the Ph.D. degree from the Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, BC, Canada, in 2018. He is currently a tenure-track Assistant Professor with the School of Big Data and Software Engineering, Chongqing University, Chongqing, China, and the Head of the Institute of Intelligent Software and Services Computing associated with Key Laboratory of Dependable Service Computing in Cyber Physical Society (Chongqing University), Education Ministry, China. He is also an adjunct research fellow of Haihe Laboratory of Information Technology Application Innovation, Tianjin, China. His current research interests are 5G/6G mobile Internet, mobile edge computing and caching, big data analytics and machine learning.

Xu Zhang (Member, IEEE) received the BS degree in communication engineering from the Beijing University of Posts and Telecommunications, China, in 2012 and the Ph.D. degree in computer science from the Department of Computer Science and Technology, Tsinghua University, China, in 2017. He is currently a Marie Sklodowska-Curie research fellow with the College of Engineering, Mathematics and Physical Sciences, University of Exeter, Exeter, U.K. His research interests include articial intelligence, multimedia communication, and network measurement. He was the co-recipient of 2019 IEEE Broadcast Technology Society Best Paper Award.

Geyong Min (Member, IEEE) received the BSc degree in computer science from the Huazhong University of Science and Technology, China, in 1995, and the Ph.D. degree in computing science from the University of Glasgow, U.K., in 2003. He is currently a professor of high-performance computing and networking with the Department of Computer Science within the College of Engineering, Mathematics and Physical Sciences, University of Exeter, U.K. His research interests include computer networks, wireless communications, parallel and distributed computing, ubiquitous computing, multimedia systems, modelling, and performance engineering.

Yongqiang Lyu (Member, IEEE) received the B.S. degree in computer science from Xidian University, Xian, China, in 2001, and the M.S. and Ph.D. degrees in computer science from Tsinghua University, Beijing, China, in 2003 and 2006, respectively. He is currently an Associate Professor with the National Research Center for Information Science and Technology, Tsinghua University. His research interests focus on processor hardware security, computer system security, networking, and the IoTs.