DRL-BASED COLLABORATIVE EDGE CONTENT REPLICATION WITH POPULARITY DISTILLATION

Haopeng Yan, Zeming Chen*, Zhi Wang† and Wenwu Zhu‡*

*,‡Department of Computer Science and Technology, Tsinghua University
†Tsinghua Shenzhen International Graduate School, †Peng Cheng Laboratory
*{yhp18, czm20}@mails.tsinghua.edu.cn, †wangzhi@sz.tsinghua.edu.cn, ‡wwzhu@tsinghua.edu.cn

ABSTRACT

The infrastructure for multimedia content delivery has been using more and more edge infrastructure (e.g., base stations, smart routers, etc.), which not only alleviates the centralized servers but also improves the quality of service by letting users access content nearby. Algorithms based on deep reinforcement learning (DRL) have been widely adopted by such edge cache replacement strategies due to their capability to adapt to changing request patterns. However, a DRL cache replacement agent learns extremely slow at an edge cache because of the sparse requests. In this paper, we propose a popularity distillation framework that allows edge caches to refer to content replication strategies of other edge caches. First, we design a collaborative edge cache framework that lets edge caches learn their strategies by handling the local requests using deep reinforcement learning and learn from others by exchanging the "soft" popularity distributions experienced by different edge caches. Second, we design a neighbor maintenance mechanism in which an agent iteratively selects only a small number of neighboring edge caches to perform the collaboration. Experiments driven by a real-world mobile video dataset show that our design can improve the cache hit rate by 3.0% compared with a non-popularity distillation baseline with only a small overhead of transmission data during distillation.

Index Terms— Edge caching; Joint strategy; Deep reinforcement learning; Knowledge distillation;

1. INTRODUCTION

With the adoption of edge-network technologies (e.g., 5G), content delivery infrastructure is undergoing an enormous change from conventional centralized to edge-based paradigm. By letting users download content from nearby caching points (e.g., base stations), edge content delivery significantly alleviates the network backbone's load. Due to both the cost-benefit and reduction of access latency, today, more and more CDN providers are using edge caches to satisfy the ever-increasing content volume. By 2021, low-latency real-time communications and high-definition video applications will leverage the multi-access edge enabled by 5G and Wi-Fi 6 and a half of all workloads will run outside the data center, either in cloud/non-cloud data centers or at the network edge [1]. Content replication, i.e., assigning content to different edge cache nodes with only limited storage and bandwidth capacities, is the key for edge content delivery to provide good quality of service for users.

Conventional solutions cannot sufficiently handle edge content replication. On the one hand, traditional "passive" cache replacement strategies, including classic LRU and LFU, whose effectiveness is based on the assumption of stability of user requests and popularity patterns [2], cannot perform well because of the dramatic change of the local popularity at the edge node, as illustrated in Fig. 1a. A more fundamental explanation is an edge cache only aware of its local requests does not have enough information to generate a replication strategy adaptive enough for future requests, especially when the local content request patterns are highly violating.

On the other hand, a centralized mechanism, which generates content replication strategies with content request information collected from all the edge caches, is more capable to "predict" future requests, but yet unacceptable due to the significant network cost and strategy deployment delay, as illustrated in Fig. 1b.

To tackle these problems, we propose collaborative edge content replication based on a *popularity distillation* approach, as illustrated in Fig. 1c. "Popularity Distillation" is our proposed design based on knowledge distillation [3], to enable edge cache nodes to exchange their own "understand" of local popularity patterns.

Basically, an edge node collects its local request records, generates the content replication strategy through a *deep reinforcement learning* framework to catch up with content request patterns. To overcome the limitation of the data scarcity at some edge nodes, we design a knowledge distillation approach to let neighboring edge nodes exchange "soft losses"

†Corresponding author.

This work is supported in part by NSFC under Grant No. 61872215, Shenzhen Science and Technology Program (Grant No. RCYX20200714114523079 and JCYJ20180306174057899).

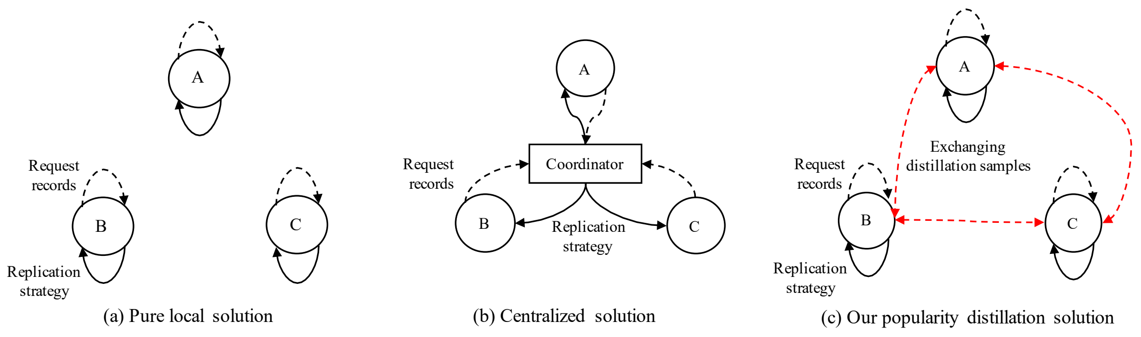


Fig. 1: Three different edge content replication solutions. (a) Pure local solution: each edge cache node generates its own strategy based on local requests; (b) Centralized solution: edge cache nodes report request records to a centralized coordinator that generate replication strategies for all edge nodes; (c) Our popularity distillation solution: edge cache node generate the content replication strategy based on both local requests and soft popularity distributions from neighbors.

of the popularity distributions and decide reference weights of other nodes by another DRL agent. The "dual" DRL framework allows an edge cache node to communicate with only a small number of edge cache nodes to achieve near centralized performance.

Our contributions can be summarized as follows.

1. We propose collaborative edge content replication that lets edge caches learn their strategies by handling their local requests and learns from others by exchanging the "soft losses". Besides, by adapting the reference weights of neighboring edge caches, an edge cache node can achieve near centralized performance by communicating only a small number of neighboring nodes.

2. To enable distillation among edge caches with diverse capacities, we design a listwise DRL cache replacement algorithm and unify the architecture of the deep model by decoupling the dimension of both the state space and the action space from cache capacity. Besides, with fewer parameters, the deep model learns faster and adapts to environmental changes more quickly.

3. We use experiments driven by a real-world mobile video dataset to show the effectiveness of our design. Our design can improve the cache hit rate by 3.0% with only 2KB overhead of information reference during distillation.

The rest of the paper is organized as follows. In Sec. 2 we review the related studies. In Sec. 3 we analyze the problem and present our detailed design of listwise DRL replacement cache. In Sec. 4 we introduce our collaborative framework of edge caches. In Sec. 5 we evaluate our design using trace-driven experiments, and we conclude the paper in Sec. 6.

2. RELATED WORKS

2.1. Content Replication

Content replication is a critical area paid much attention to by both industry and academia. The content replication strategy highly depends on the content popularity of user request patterns, which includes three types [4]: i) finite request sequences; ii) stationary request sequences; and iii) non-stationary request sequences. Conventional passive cache replacement strategies, including LRU and LFU, cannot solve the third type of request sequences well due to the non-stationary characteristics. To learn the dynamics of content popularity, the reinforcement learning(RL) method has been introduced to cache problems. The RL-based strategy can learn user patterns implicitly and have a significant improvement in cache utilization.

2.2. Deep Reinforcement Learning Cache

The past few years observe the notable progress of deep reinforcement learning(DRL). In the area of network and communication, DRL has been used as a tool to address various problems (e.g., [5, 6]). By defining a reasonable reward as the goal, many model-free DRL algorithms can be applied to solve cache problems.

However, the high dimension of the state space and the action space in the cache replacement problem leads to large neural networks, making models hard to converge. To tackle this challenge, Zhong et al. [7] proposed a DRL-based online cache replacement policy to reduce the action space. However, the reduction of the action dimension has a limited effect on reducing the size of the neural network because the state space is still proportional to the cache capacity. To solve this problem, we propose a listwise DRL cache replacement policy, whose neural network is irrelevant to capacity, significantly reducing its model size.

2.3. Collaboration in Edge Cache

Enabling edge cache collaboration is a critical practice to improve content replication performance. Wang et al. [8] propose 'MacoCache', a multi-agent DRL-based algorithm to minimize both content access latency and traffic cost by letting edge learns its policy in conjunction with other edges. In our scenario, we proposed a dual-RL architecture that incorporates knowledge distillation [3] for edge-network cache collaboration, one for cache replacement, and one for collaboration.

Table 1: Notations

Symbol	Definition
C	Cache capacity
s_t^i, a_t^i, r_t^i	State/action/reward of i-th cache unit
l_j, F	Time slide window and number of them
$\mathcal{M}$	Replay memory in DRL agent
$V_\theta(\cdot)$	Popularity value function
d_k, N	Edge cache k and number of edge caches
θ_k	Parameters of cache model in d_k
$\phi_{i,j}$	Reference weights of d_i to d_j
$\mathcal{B}_k$	Mini-batch of soft samples from d_k

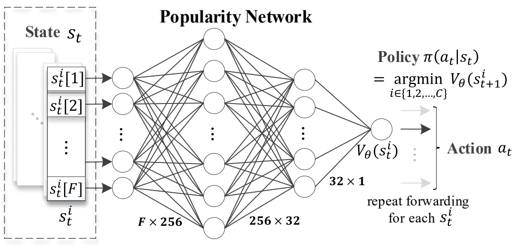


Fig. 2: LWDRL Replacement Cache Policy.

3. LISTWISE DRL-BASED REPLACEMENT CACHE

In this section, we will explain how a single edge cache node performs cache replacement through a listwise deep reinforcement learning (LWDRL) approach.

The cache replacement problem can be formulated as an MDP because the cache hit rate constitutes the natural reward. Thus, many model-free DRL algorithms can be applied to solve it. However, the state space dimensions are proportional to cache capacity C. As a result, the deep model's architecture is also related to C, which leads to large neural networks and hinders knowledge distillation among different edge devices for the unified model architectures.

Actually, the order of elements in the cache will not affect the cache's functions. However, cache units' states are arranged according to their storage order and constitute the cache state. Different arrangement orders lead to redundant cache states, making it hard for DRL models to learn.

To tackle this problem, we propose LWDRL replacement cache, as illustrated in Fig. 2, where the replacement in each cache unit is considered an MDP as presented in 3.1. Then, the state space dimension will be reduced from $C \times F$ to F, no longer relevant to C. In this way, the architecture of DRL models in edge caches are unified, laying a good foundation for knowledge distillation. Besides, the parameters of DNN are significantly reduced, and the deep model will converge more quickly.

3.1. MDP Formulation

To model cache replacement as an MDP, we need to design the state, the action, and the reward:

State design. The cache state s_t consists of a list of sub-states $(s_t^1, s_t^2, ..., s_t^C)$, where s_t^i is the state of the i-th cache unit. Each cache unit state has F elements, which means the frequencies of content in different periods in the past.

Action design. When 'miss' happens, the cache need to select which content stored in it to be evicted. The action is $a_t \in \{1, \cdots, C\}$, and it can be represented in a one-hot form: $a_t = (a_t^1, a_t^2, ..., a_t^C)$, s.t. $\sum_{i=1}^C a_t^i = 1$ where $a_t^i \in \{0, 1\}$ is the action of each cache unit.

Reward design. To maximize the cache hit rate, we use the number of 'hits' between two adjacent cache misses as the reward r_t. By counting the hits of each cache unit separately, we can get the rewards of all cache units $r_t = (r_t^1, r_t^2, ..., r_t^C)$.

Actually, each cache unit does replacement as an MDP, and we can train a model on the cache unit level.

3.2. Detailed Design

When requests arrive, the edge device handles them one by one. Whenever the content requested 'misses' in the edge cache, the cache agent must evict one content from the cache to spare storage space for new content. Sequences of 'cache misses' and replacement decisions take in turn, making up an agent-environment interaction loop.

Every time DRL agent makes decision, it takes two steps: (a) Action step: agent interacts with the environment and collects experiences. (b) Training Step: agent trains itself with experience in its replay memory. How the DRL agent does cache replacement is presented in Algo. 1, and we will explain it in detail next.

3.2.1. Action Step

Upon any 'cache miss' event, the agent observes state from environment and then estimates state values of all cache units and choose the content with smallest state value to evict, as is shown in Fig. 2. The policy is:

$$\pi_\theta(a_t|s_t) = \arg\min_{i \in \mathcal{A}}(V_\theta(s_t^i)), \tag{1}$$

where $V_\theta(\cdot)$ is the popularity network.

After the environment taking the action, the agent observe new state s_{t+1} and reward r_t. For each agent-environment loop, the agent collects a transition (s_t, a_t, r_t, s_{t+1}) and saves it in its replay memory $\mathcal{M}$.

3.2.2. Training Step

Then, the agent samples a mini-batch of transitions $\mathcal{B} = \{(s, a, r, s')\}$ from its replay memory $\mathcal{M}$ and trains with them by minimizing MSE loss:

$$\mathcal{L}(\mathcal{B}|\theta) := \frac{1}{\|\mathcal{B}\|} \sum_{(s,a,r,s') \in \mathcal{B}} \sum_{i=1}^{C} \left[V_\theta(s^i) - y^i\right]^2$$
$$\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta),\ \bar{\theta} \leftarrow \lambda\theta + (1 - \lambda)\bar{\theta} \tag{2}$$

where $y^i := r^i + \gamma V_{\bar{\theta}}(s'^i)$ is the target state value.

By learning the optimal state value function via value iteration, we can get the optimal policy.

Algorithm 1 List-wise DRL-based Cache Replacement

```
1: $\theta \sim$ some initialization distributions, $\bar{\theta} \leftarrow \theta$, $\mathcal{M} \leftarrow \emptyset$
2: Observe initial state $s_t$
3: for t = 1, 2, ..., $K$ do
4:     Select action $a_t := \arg\min_{i \in \mathcal{A}}(V_\theta(s_t^i))$.
5:     Execute action $a_t$ in cache environment and observe reward $r_t$ and new state $s_{t+1}$.
6:     Store transition $\mathcal{M} \leftarrow \mathcal{M} \cup \{(s_t, a_t, r_t, s_{t+1})\}$
7:     if $t \mod T_u == 0$ then
8:         Sample transitions $\mathcal{B} := \{(s, a, r, s')\} \sim \mathcal{M}$
9:         Update popularity network according to Eq. 2.
10:    end if
11:    if $t \mod T_d == 0$ then
12:        Execute popularity distillation in Algo. 2.
13:    end if
14: end for
```

4. COLLABORATION BY POPULARITY DISTILLATION

In this section, we will describe the details of the popularity distillation, as is illustrated in Algo. 2. The collaboration process through popularity distillation consists of two stages: (a) generating distillation samples; (b) exchanging samples and training.

4.1. Distilling Popularity Samples

In this stage, firstly, the DRL agent samples transitions from its replay memory $\mathcal{T}_k = \{(s, a, r, s')\} \sim \mathcal{M}$. In each transition, the difference between two adjacent states is the cache unit to be replaced, which can be located by the action, as is shown in Fig. 3b. Then we can distill samples:

$$\mathcal{B}_k = \{(s^a, V_{\theta_k}(s^a)), (s, a) \sim \mathcal{T}_k\} \tag{3}$$

4.2. Learning by Popularity Distillation

When distillation samples prepared, each agent first selects η edge caches $\mathcal{N}_k$ randomly and pulls samples from them $\mathcal{X}_k = \{\mathcal{B}_j, j \in \mathcal{N}_k\}$. Then agent trains itself with these samples by minimizing the weighted sum of MSE losses:

$$\begin{aligned} \mathcal{L}_{PD}(\mathcal{X}_k|\theta_k, \phi_k) &= \frac{1}{\eta} \sum_{\mathcal{B}_j \in \mathcal{X}_k} w_k^j \mathcal{L}(\mathcal{B}_j|\theta_k) \\ \mathcal{L}(\mathcal{B}_j|\theta_k) &= \frac{1}{\|\mathcal{B}_j\|} \sum_{x,y \sim \mathcal{B}_j} [V_{\theta_k}(x) - y]^2 \end{aligned} \tag{4}$$

where $w_k^j = \frac{e^{\phi_k^j}}{\sum_i e^{\phi_k^i}}$ is the reference weight and $\mathcal{L}(\mathcal{B}_j|\theta_k)$ is the loss value of the samples from d_j in the model of d_k.

4.3. Adaptive Reference Weights

Since the environments in the edge network are diverse and changing dynamically, an edge cache should not refer to others equally and statically. Thus, we use another DRL agent to decide the weights refer to others. Basically, an edge device maintains a set of reference weights to other devices. Every time taking popularity distillation, it will update both its agent parameters and its reference weights.

Algorithm 2 Popularity Distillation in d_k with $\{\theta_k, \phi_k, \psi_k\}$

```
1: Generate PD samples $\mathcal{B}_k$ according to Sec. 4.1.
2: Randomly select $\eta$ neighbors $\mathcal{N}_k$ and pull PD samples from them: $\mathcal{X}_k = \{\mathcal{B}_j, j \in \mathcal{N}_k\}$.
3: Caculate losses of these samples in replacement model in $d_k$ as the state: $s_t = \{\mathcal{L}(\mathcal{B}_j|\theta_k), j \in \mathcal{N}_k\}$.
4: Mixing the losses above through the policy of reference agent: $\boldsymbol{a_t \sim \mu_{\phi_k}(a_t|s_t) = \mathcal{L}_{PD}(\mathcal{X}_k|\theta_k, \phi_k)}$.
5: Execute action $a_t$ in environment (update replacement model) and observe reward $r_t$ and new state $s_{t+1}$:
   $\boldsymbol{\theta_k \leftarrow \theta_k - \alpha \nabla_{\theta_k} \mathcal{L}_{PD}(\mathcal{X}_k|\theta_k, \phi_k)}$
6: Store transiton: $\mathcal{M} \leftarrow \mathcal{M} \cup \{(s_t, a_t, r_t, s_{t+1})\}$
7: Sample transitions $\{(s, a, r, s')\}$ from replay memory $\mathcal{M}$ and update the critic according to Eq. 6, and update the actor according to Eq. 7.
```

First, we define the process of mixing losses of samples from other edge devices as a decision-making process and optimize the reference weights through DDPG [9], which consists of an actor and a critic. The actor acts as follows:

$$\mu_{\phi_k}(a|s) = \mathcal{L}_{PD}(\mathcal{X}_k|\theta_k, \phi_k) = \frac{1}{\eta} \sum_{j \in \mathcal{N}_k} w_k^j \mathcal{L}_t^j, \tag{5}$$

where the state $s = \{\mathcal{L}^j, j \in \mathcal{N}_k\}$ is made up of the losses of samples from other edge devices. Every time agent interacts with the environment, it collects experience and saves it as a transition to its replay memory. Periodically agent samples a mini-batch of transitions and updates both the critic and the actor with them. The DRL agent updates the critic as follows:

$$\begin{aligned} J(\psi_k) = E_{(s,a,r,s') \sim \mathcal{M}}[r + \gamma Q_{\psi_k}(s', \mu_{\phi_k}(s')) \\ - Q_{\psi_k}(s, a)]^2 \\ \psi_k \leftarrow \psi_k - \alpha \nabla_{\psi_k} J(\psi_k) \end{aligned} \tag{6}$$

And then update the actor by policy gradient:

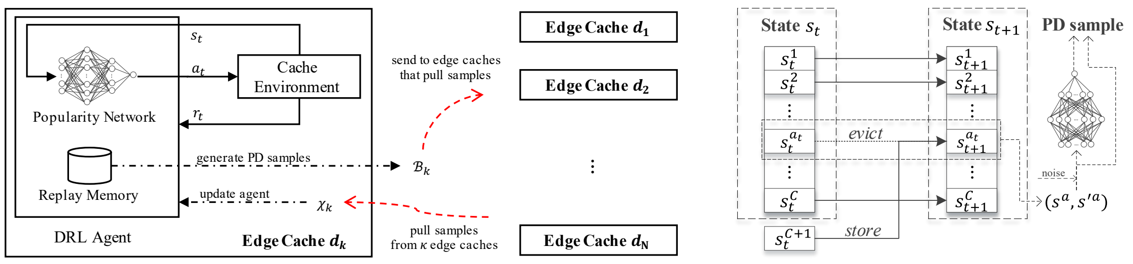
$$\begin{aligned} \nabla_{\phi_k} J(\phi_k) &= -Q_{\psi_k}(s, \mu_{\phi_k}(s)) \nabla_{\phi_k} \mu_{\phi_k}(s)) \\ &= -Q_{\psi_k}(s, \mu_{\phi_k}(s))(\frac{1}{\eta} \sum_{j \in \mathcal{N}_k} w_k^j (1 - w_k^j) \mathcal{L}^j \nabla_{\phi_k} \phi_k) \\ \phi_k &\leftarrow \phi_k - \alpha \nabla_{\phi_k} J(\phi_k) \end{aligned} \tag{7}$$

By updating the critic and the actor in turn, an edge node can learn the proper reference weights (policy) to other nodes.

5. EVALUATION

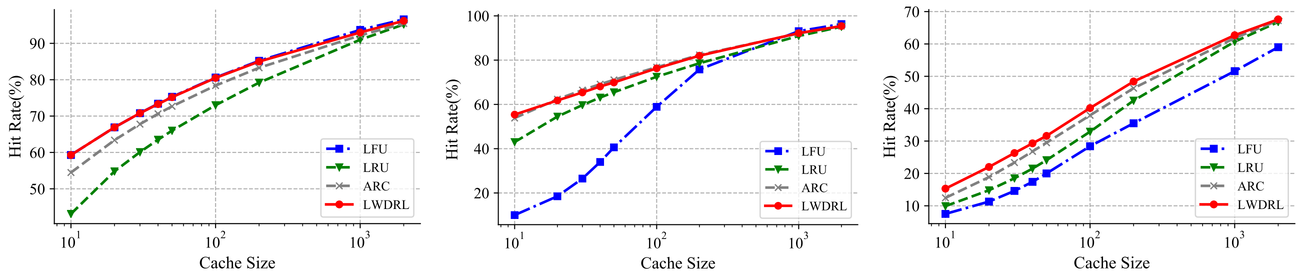
5.1. Dataset

The real-world dataset we used is from iQiYi company, one of China's largest video service providers. This dataset



(a) Edge cache exchanges distillation samples with others. (b) Generating distillation sample.

Fig. 3: Collaboration for DRL-based caches.



(a) Static Zipf dataset (b) Dynamic Zipf dataset (c) IQiYi PoIs dataset

Fig. 4: Hit rate of different algorithms in single edge cache.

includes about 53 million content requests in Beijing for two weeks, and each request consists of user ID, timestamp, location, and content name. In our experiments, we choose several points of interest (PoIs) and filter the requests in them for training and testing. To simulate the sparsity of requests on different edge caches, we randomly discard a portion of the requests according to the sparsity ratio ρ.

To evaluate algorithms' performance under IRM, we generate datasets where the content popularities subject to Zipf distributions. In the static Zipf dataset, the content popularities are stationary. For the dynamic Zipf dataset, the content popularities are changing periodically by permuting the content ranks. To avoid dramatic changes between two adjacent periods, we limit rank shifts to no more than 10.

5.2. Experiment Setup

notation	value	notation	value
γ	0.99	η	3
T_u	2	T_d	100
α	0.0003	ρ	0.2

Table 2: Experiment parameter settings

The baselines include traditional cache replacement algorithms like LRU, LFU, ARC [10].

For deep learning methods, the features we used are the content frequencies in different periods in the past. The lengths of time windows for counting frequencies are 10 seconds, 1 minute, 5 minutes, 30 minutes, 1 hour, one day, two days, and one week.

In every distillation period (10 second), the overhead of distillation is $2 \times (F+1) \times \|\mathcal{B}\| \times \eta$, and when batch size $\|\mathcal{B}\|$=32, F=8, η=4, the overhead is about 1.73KB. The other parameter settings are listed in Tab. 2.

To utilize the cache, we use the cache hit rate as the metrics for evaluation. In evaluation, all algorithms make decisions and learning online. In other words, they will only go through the dataset once instead of repeatedly learning and fitting on the same dataset.

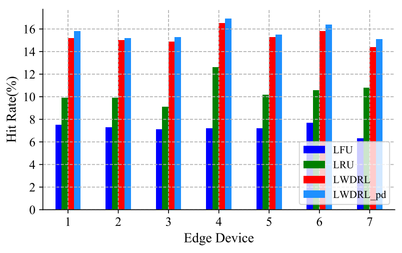
5.3. Performance Evaluation

5.3.1. Evaluation in Single Cache

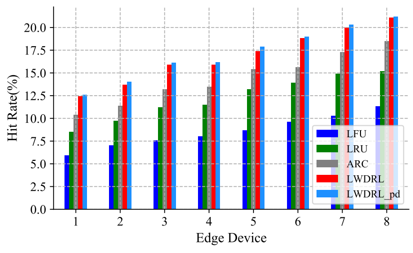
Fig. 4a shows the results in the static zipf dataset, where the content popularities are stationary. In this situation, LFU is considered optimal. Compared with LFU, our algorithm can catch the content popularities quickly and achieve near-optimal results (99.3% of LFU).

Fig. 4b shows the results in the dynamic zipf dataset, where content popularities are non-stationary and changing as time passes by. As a result, LFU performs poorly. In contrast, LRU and ARC can handle this situation well because they will not be influenced by history a long time ago while ARC can balance the weights of different periods, so it outperforms LRU. In DRL-based algorithms, the agent can also learn the weights of different periods of the past, so it works as well as ARU.

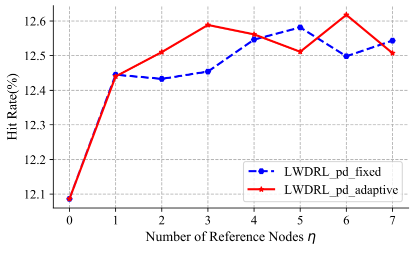
Fig. 4c shows the results of algorithms in the iQiYi dataset, where the dynamics of content popularities are more complex than two Zipf datasets. In this situation, our algo-



(a) Edge caches with same capacity(10).



(b) Edge caches with different capacities (10, 10, 15, 15, 20, 20, 25, 25).



(c) Average hit rates in the configurations with different number of reference nodes

rithm performs best than others and has at most 22.8% improvement compared with ARC.

5.3.2. *Evaluation in Multi-cache Environment*

In the multi-cache environment, when capacity is unified, as presented in Fig. 5a, the popularity distillation contributes to improving the hit rate by 3.0% on average. Fig. 5b shows the result of edge caches with different capacities. Popularity distillation works in edge caches with diverse capacities and has improved cache hit rate by about 1.6% on average. The effectiveness of distillation decreases as capability increases.

For LWDRL cache, it learns the content popularities with dynamic experience in the transition from one cache unit state to another, and this transition occurs only when cache replacement happens. The 'dynamic border' between two sub-states is related to cache capacity. Thus, the dynamic experience from the large cache can improve popularity evaluation accuracy in the small one. In contrast, the dynamic experience from the small one has slight improvement for the large one because of the 'border' of dynamic of small cache often within that large cache. Thus, adaptive weight reference allows nodes to update the reference weights to avoid biased samples from interfering with learning.

Fig. 5c shows the influence of the number of reference nodes. As shown in the figure, the hit rate increases as η increases initially and has little improvement when $\eta > 3$. In our experiments, we use $\eta = 3$ for training and testing.

6. CONCLUSION AND FUTURE WORK

In conclusion, we first design a listwise DRL-based replacement cache that decouples the deep model from cache capacity and keeps models unified in all edge caches. Then, we design a popularity distillation mechanism letting edge nodes refer others by exchanging distillation samples and learns the reference weights of neighboring edge caches. Our experiments show that with popularity distillation, an edge cache node can achieve near centralized performance by communicating with only a small number of neighboring nodes.

7. REFERENCES

[1] U Cisco, "Cisco annual internet report (2018–2023) white paper," 2020.

[2] Ge Ma, Zhi Wang, Miao Zhang, Jiahui Ye, Minghua Chen, and Wenwu Zhu, "Understanding Performance of Edge Content Caching for Mobile Video Streaming," in *IEEE Journal on Selected Areas in Communications (JSAC)*, 2017.

[3] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[4] Georgios S Paschos, George Iosifidis, Meixia Tao, Don Towsley, and Giuseppe Caire, "The role of caching in future communication systems and networks," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 6, pp. 1111–1125, 2018.

[5] Vadim Kirilin, Aditya Sundarrajan, Sergey Gorinsky, and Ramesh K Sitaraman, "Rl-cache: Learning-based cache admission for content delivery," 2019.

[6] Ye Jiahui, Li Zichun, Wang Zhi, Zheng Zhuobin, Hu Han, and Zhu Wenwu, "Joint cache size scaling and replacement adaptation for small content providers," in *Proceedings of IEEE INFOCOM 2021, Virtual Conference, China, May*, 2021, pp. 10–13.

[7] Chen Zhong, M Cenk Gursoy, and Senem Velipasalar, "A deep reinforcement learning-based framework for content caching," in *2018 52nd Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 2018, pp. 1–6.

[8] Fangxin Wang, Feng Wang, Jiangchuan Liu, Ryan Shea, and Lifeng Sun, "Intelligent video caching at network edge: A multi-agent deep reinforcement learning approach," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 2499–2508.

[9] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[10] Nimrod Megiddo and Dharmendra S Modha, "Arc: A self-tuning, low overhead replacement cache.," in *Fast*, 2003, vol. 3, pp. 115–130.