Joint Cache Size Scaling and Replacement Adaptation for Small Content Providers

Jiahui Ye[1†], Zichun Li[1†], Zhi Wang[2*], Zhuobin Zheng[3], Han Hu[4], and Wenwu Zhu[3*]
[1]Tsinghua-Berkeley Shenzhen Institue, Tsinghua University
[2]Shenzhen International Graduate School, Tsinghua University & Peng Cheng Laboratory
[3]Department of Computer Science and Technology, Tsinghua University
[4]School of Information and Electronics, Beijing Institute of Technology
{yejh16, li-zc18, zhengzb16}@mails.tsinghua.edu.cn,
wangzhi@sz.tsinghua.edu.cn, hhu@bit.edu.cn, wwzhu@tsinghua.edu.cn

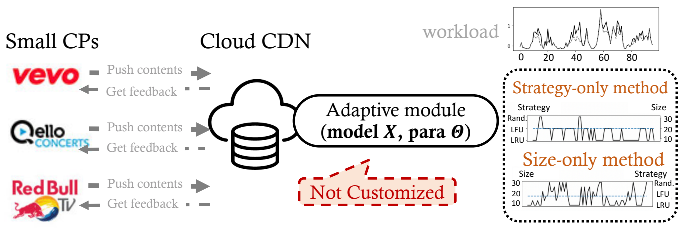
***Abstract*—Elastic Content Delivery Networks (Elastic CDNs) have been introduced to support explosive Internet traffic growth by providing small Content Providers (CPs) with just-in-time services. Due to the diverse requirements of small CPs, they need customized adaptive caching modules to help them adjust the cached contents to maximize their long-term utility. The traditional adaptive caching module is usually a built-in service in a cloud CDN. They adaptively change cache contents using size-scaling-only methods or strategy-adaptation-only methods. A natural question is: can we jointly optimize size and strategy to achieve tradeoff and better performance for small CPs when renting services from elastic CDNs? The problem is challenging because the two decision variables could involve both discrete and categorical variables, where *discrete variables have an intrinsic order while categorical variables do not*. In this paper, we propose a distribution-guided reinforcement learning framework JEANA to learn the joint cache size scaling and strategy adaptation policy. We design a distribution-guided regularizer to keep the intrinsic order of discrete variables. More importantly, we prove that our algorithm has a theoretical guarantee of performance improvement. Trace-driven experimental results demonstrate our method can improve the hit ratio while reducing the rental cost.**

***Index Terms*—elastic content delivery networks, caching, deep reinforcement learning.**

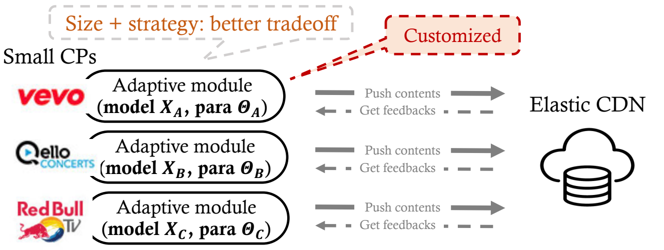
I. Introduction

Recent years have witnessed a rapid rise of small content providers: many vertical video-sharing platforms have to operate their video streaming due to their unique interactive requirements (*e.g.,* online education platforms need to incorporate quiz in video streaming) that cannot be well handled by traditional large video sharing platforms (*e.g.,* YouTube). Such small content providers can attract a large number of users and drive significant internet traffic (*e.g.,* Coursera had over 30 million users worldwide by the end of 2018).

Comparing with traditional large content providers that maintain their self-built CDNs, small content providers tend to lease resources from the emerging elastic CDNs (*e.g.,* Akamai Aura [1] and Huawei uCDN [2]), for their content delivery. Unlike the Cloud CDN, which has a built-in adaptive cache module that automatically scales the rental cache size for tenants, elastic CDN [1] provides tenants API to determine what contents to cache. This gives tenants a customized choice to employ their own model and parameters. This application raises a problem: how to maximize utility for small content providers in elastic CDNs through adaptive caching?



(a) Traditional adaptive caching solutions in cloud CDN.


Size + strategy: better tradeoff
Customized
Small CPs
vevo
Adaptive module (model X_A, para Θ_A)
Qello CONCERTS
Adaptive module (model X_B, para Θ_B)
Red Bull TV
Adaptive module (model X_C, para Θ_C)
Push contents
Get feedbacks
Elastic CDN

(b) Our joint adaptive caching solution for small content providers.

Fig. 1: Comparison between previous studies on adaptive caching and our joint adaptive caching solution.

Existing adaptive caching solutions usually attach to service providers (*e.g.,* self-built CDN or cloud CDN). These solutions can be categorized into two groups: adaptively scaling the cache size (size only) [3]–[5], or adaptively adjusting the cache replacement strategy (strategy only) [6]–[8]. A strategy-only method can lead to higher hit rates but can introduce higher strategy switching overhead costs. Replacing old contents with new ones will cause costs due to data migration. Different strategies usually prioritize different contents. When switching to a new strategy, we may need to replace a set of contents at once and have a high switching overhead cost. Although a size-only method does not bring switching costs, it is

[†]Jiahui Ye and Zichun Li are co-first authors. [*]Zhi Wang and Wenwu Zhu are corresponding authors.

not able to adapt to content popularity changes, and thus can not guarantee a high cache hit rate. Therefore, when designing adaptive caching policy for small content providers, a natural question is: *can we jointly scale size and adjust replacement strategy?* A joint solution is capable of trading off between cache hit rate and strategy switching costs to maximize tenants' benefits.

We now need to solve this joint adaptive caching problem which has a huge search space and no prior knowledge on future content popularities. In recent years, deep reinforcement learning (DRL), which learns a policy via interacting with the environment, has achieved great success in various kinds of applications. Recently, there are some works that use DRL to make content replacement strategy. They are all single-action frameworks focusing on cache admission or prefetch and usually assume the arrival process as an i.i.d. model [9]. [10] used DQN to achieve caching strategy adaption. Although DQN is able to deal with large continuous state space, it suffers from low convergence when dealing with multi-dimension action space. [11] added Wolpertinger to DDPG to deal with large binary action space. But in our problem, our actions are discrete variables or categorical variables, or both. Existing off-the-shelf DRL algorithms treat discrete and categorical cases equally as classification tasks, ignoring an important fact that *discrete variables have intrinsic order.* Thus, directly applying DRL algorithms will cause performance degradation. (Details are given in Sec. III-C)

The main contributions of this paper are as follows:

- To simultaneously optimize both categorical and discrete variables in joint adaptive caching problem, we propose a novel RL algorithm JEANA to learn a joint policy. It utilizes a distribution-guided regularizer based on Wasserstein distance to constrain the action distribution to be smoother and capture intrinsic order among discrete variables, which improves model representation and capacity.
- We prove that our algorithm has theoretical guarantee of monotonic performance improvement by maximizing a lower bound of expected reward at each iteration.
- We evaluate JEANA on real video-on-demand content request traces. Experimental results show: 1) the superiority of joint caching decision over strategy-only and size-only methods in terms of cache hit rate and rental cost; 2) the stability and performance improvement of our training process; 3) the effectiveness of distribution-guided regularizer in reducing performance loss.

The remainder of this paper is organized as follows. The system model and problem definition are introduced in Sec. II. Our proposed JEANA framework is presented in Sec. III. Sec. IV discusses the experiment results. We review the related work in Sec. V and conclude the paper in Sec. VI.

II. System Model

In this section, we first present the workflow of JEANA. It shows how JEANA works in the elastic CDNs. Then we elaborate on the system model and give our problem definition.

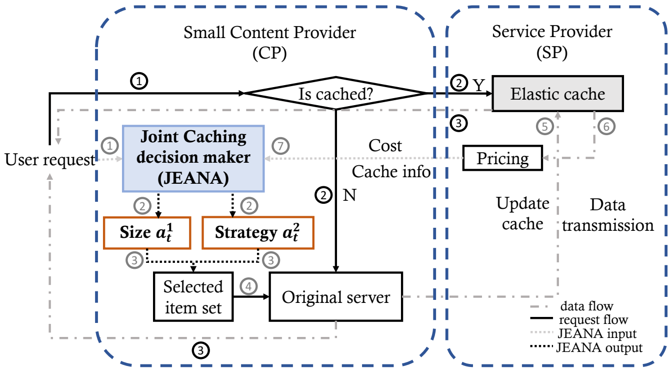


Fig. 2: The workflow of JEANA. The light blue block represents the JEANA module that locates at the CP side.

A. Workflow

As shown in Fig. 2, when receiving a user request, the CP checks whether the content is cached in the elastic cache of the SP. If yes, the request will be served directly through the cache; otherwise, it will be directed to the original server. These processes are represented by black circles. At the same time, the user request will also be input to JEANA. JEANA outputs the joint action, namely the cache size scaling variable and strategy adaptation variable. According to these two variables, we derive the indices of contents that currently need to be cached. Then the original server pushes these contents to the elastic cache. SP charges the current cache lease according to its own pricing mechanism. SP will send the cost and current cache status info as feedback to JEANA. These processes are represented by gray circles. In summary, JEANA learns and updates policy based on user requests and SP feedbacks.

B. Use Cases

We define *categorical control* and *discrete control* to illustrate that the two decision variables may be different types.

Use case 1 (categorical control case): A small CP C_A provides video service for its users. C_A needs to rent elastic CDNs service from SP S_B. The contents of C_A are equal-size chunks, whose size equal to cache unit size of S_B. At every decision round $t \in \{0, 1, 2, \cdots, T-1\}$, C_A needs to decide how many cache units to rent from S_B and what contents to cache. C_A selects one of the three cache replacement strategies of LRU, LFU, and Random, represented by a_t^2, to rank contents, and select the top a_t^1 items to cache. The size scaling variable a_t^1 is a discrete variable, and there is intrinsic order between the values. Strategy adaptation variable a_t^2 represents a categorical variable, and there is no intrinsic order between the values.

Use case 2 (discrete control case): The contents of C_A are equal-size chunks, whose size equal to cache unit size of S_B. C_A needs to decide how many cache units to rent from S_B and what contents to cache. The cache replacement strategy adopted by C_A is a modified ALRFU [6], which can shift the cache strategy combining frequency and recency through a parameter $\delta_t \in \{0, 0.1, 0.2, \cdots, 1\}$. This dynamic scoring mechanism considers the size, frequency, and recency. So at every decision round t, C_A selects a value a_t^2 from the parameter set, uses the corresponding strategy to rank the contents by their scores, and select top a_t^1 items to cache. The size scaling variable a_t^1 and strategy adaptation variable a_t^2 are both discrete variables, they both have intrinsic order.

C. Problem Definition of Joint Elastic Caching

In this paper, we consider the following scenario: a small CP owns a library of equal-sized contents $f \in \{1, 2, \cdots, F\}$. This small content provider needs to rent caches from a third-party SP to deliver its contents effectively. The SP provides paid elastic caching service, and it provides tenants API to determine what contents to cache. The CP can determine what contents to cache by jointly scaling cache size and adjusting replacement strategy in a slotted fashion during a finite horizon T. We denote the cache size scaling decision at time t as a_t^1, and cache replacement strategy adaptation decision at time t as a_t^2. The joint decision at time t is denoted as $\mathbf{a}_t = [a_t^1, a_t^2]$. SP charges CP for renting caches every timeslot. Note that the dynamic cache leasing price is agnostic to decision maker.

For small CPs, they aim to maximize the gains from cache hits while minimizing the rental cost over a given time horizon. We use a common form of utility function to define their goal:

$$U = \sum_{t=0}^{T-1} (g_t - d_t) = \sum_{t=0}^{T-1} (\lambda h_t - d_t) \tag{1}$$

The utility comes from the gain g_t of serving content requests by cache minus the rental cost d_t on leasing cache storage, i.e., $g_t - d_t$. When requested content is found in the SP's cache, and this is referred to as a "cache hit". When requested content is not found in the cache and must be fetched from a remote CP's original server, this is referred to as a "cache miss" and will result in high latency. Obviously, every cache hit brings gain. We denote λ as per cache hit gain. At each timeslot, the total hits is denoted as h_t, and total cache hit gain is denoted as $g_t = \lambda h_t$. Rental cost at each timeslot is denoted as d_t. Thus, utility $u_t = g_t - d_t = \lambda h_t - d_t$. d_t and h_t are directly related to which contents are cached. That is, d_t and h_t are determined by joint caching action $\mathbf{a}_t = [a_t^1, a_t^2]$. Table I summarizes notations used in this paper.

Our goal is to find the policy π^* that maximizes the CP's utility over time horizon T. The mathematical expression of the joint adaptive caching problem is as follows:

$$\pi^* = \arg\max_{a^1, a^2} U. \tag{2}$$

III. JEANA Framework

In this section, we first depict the overall framework of JEANA followed by detailed descriptions and analysis of each block: i) MDP definition for DRL, ii) an actor-critic network for generating joint action, and iii) loss function. To reduce the performance loss when dealing with different types of action variables, we design a Wasserstein distance based regularizer to preserve intrinsic order between discrete variables. This regularizer improves model representation and capacity of neural networks. Then we derive a new objective function. Next, we summarize the whole algorithm. Last, we prove that our algorithm has theoretical guarantee of monotonic performance improvement based on the top of TRPO [13].

TABLE I: Notation Definition

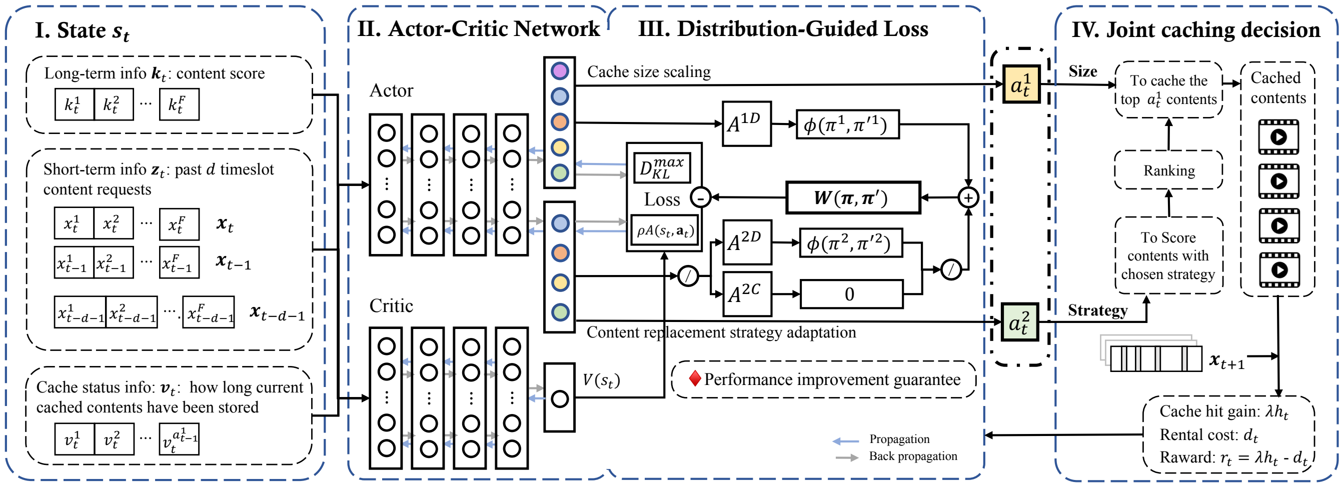
Parameter	Definition
F	the number of contents owned by CP
T	time horizon
λ	revenue per cache hit
$\mathbf{k_t}$	long-term content scores at time t
$\mathbf{z_t}$	short-term content scores at time t
$\mathbf{v_t}$	cache status information at time t
h_t	number of cache hits at time t
g_t	cache hits gain at time t
d_t	cost of leasing cache from SP at time t
r_t	utility/reward of time t, $r_t = g_t - h_t$
δ_t	parameter combines recency and frequency
x_t^f	request number of content f at time t
U	utility function of joint caching desicion
$\mathbf{R}_t$	requests vectors received by CP at time t
Variable	Definition
a_t^1	cache leasing size at time decison
a_t^2	cache replacement strategy decision
$\mathbf{a}_t$	joint caching decision, $\mathbf{a}_t = [a_t^1, a_t^2]$
π^i	sub-policy of action $a^i, i \in \{1, 2\}$
π	policy generates the joint action $\mathbf{a}$

A. MDP Tuple

A finite-horizon Markov Decision Process (MDP) is defined by a tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the transition probability distribution, $r : \mathcal{S} \to \mathbb{R}$ is the reward function, and $\gamma \in (0, 1)$ is the discount factor.

First, let us define state, action, and reward for our problem. In our setting, state transition occurs upon every timeslot.

State s_t consists of three parts: long-term information $\mathbf{k}_t$, short-term information $\mathbf{z}_t$ and cache status information $\mathbf{v}_t$. Thus, $s_t = \{\mathbf{k}_t, \mathbf{z}_t, \mathbf{v}_t\}$. $\mathbf{k}_t$ is the score vector that contains score values for each content at time t. It is determined by the replacement strategy we choose and the corresponding scoring mechanism. In discrete control case, $\mathbf{k}_t = \delta_t \mathbf{k}_{t-1} + \mathbf{x}_t$. A control parameter δ_t is used to make a tradeoff between recency and frequency. In this way, we are able to take advantage of both LRU and LFU [6]. By modifying δ_t, we can adapt to non-stationary traffic conditions, i.e., when the popularity of content evolves with time. $\mathbf{x}_t$ stands for a request vector that records the number of requests for each content at time t. δ_t is adaptively determined by the replacement strategy adaption action variable a^2. If $\delta_t \equiv 1$ for all $t \in \{0, 1, 2, \cdots, T-1\}$, $\mathbf{k}_t = \sum_{i=0}^{t} x_i$ and it degenerates to the frequency count; so the replacement will simply become the LFU strategy. Similarly, if $\delta_t \equiv 0$ for all $t \in \{0, 1, 2, \cdots, T-1\}$, $\mathbf{k}_t = \mathbf{x}_t$, and it degenerates to the recency count. Short-term information $\mathbf{z}_t$ records content request vectors of the past d timeslots. We denote $\mathbf{z}_t = \{\mathbf{x}_t, \mathbf{x}_{t-1}, \cdots, \mathbf{x}_{t-d-1}\}$. Cache status information $\mathbf{v}_t$ records how many consecutive timeslots the current cached contents have been stored, that is, $\mathbf{v}_t = \{v_t^0, v_t^1, \cdots, v_t^{a_{t-1}^1}\}$.



I. State s_t
Long-term info k_t: content score
Short-term info z_t: past d timeslot content requests
Cache status info: v_t: how long current cached contents have been stored
II. Actor-Critic Network
Actor
Critic
$V(s_t)$
III. Distribution-Guided Loss
Cache size scaling
Content replacement strategy adaptation
D_{KL}^{max}
Loss
$\rho A(s_t, a_t)$
$W(\pi, \pi')$
$\phi(\pi^1, \pi'^1)$
$\phi(\pi^2, \pi'^2)$
A^{1D}
A^{2D}
A^{2C}
Performance improvement guarantee
Propagation
Back propagation
IV. Joint caching decision
Size
Strategy
To cache the top a_t^1 contents
Ranking
To Score contents with chosen strategy
Cached contents
x_{t+1}
Cache hit gain: λh_t
Rental cost: d_t
Reward: $r_t = \lambda h_t - d_t$

Fig. 3: Overall framework of JEANA. The left part state s_t is the input of the actor-critic network. s_t contains long-term information $\mathbf{k}_t$, short-term information $\mathbf{z}_t$ and cache status $\mathbf{v}_t$. The middle part consists an actor-critic network and its loss function, which contains a distribution-guided loss $W(\pi, \pi')$. The actor network learns a joint caching decision $\mathbf{a}_t = [a_t^1, a_t^2]$, and the critic network outputs state value $V(s_t)$. If action a_t^* belongs to discrete action space $\mathcal{A}^{*\mathcal{D}}$, then we add a Wasserstein distance term $\phi(\pi^*, \pi'^*)$ into $W(\cdot)$; otherwise, a_t^* belongs to categorical action space $\mathcal{A}^{*\mathcal{C}}$, and we add 0 into $W(\cdot)$. The right part shows how joint caching decision $\mathbf{a}_t = [a_t^1, a_t^2]$ determines what contents to cache, and how much reward r_t it achieves.

Action $a_t = [a_t^1, a_t^2]$. a_t^1 is a scalar that indicates cache size scaling variable, a_t^2 is a scalar that indicates the replacement strategy adaptation variable.

Reward r_t. We define r_t as utility. As described in Sec. II-C, we define reward as $r_t = g_t - d_t = \lambda h_t - d_t$. Thus, the total utility of joint adaptive caching decision is $U = R = \sum_{t=0}^{T} r_t$.

B. Actor-Critic Network and Joint Caching Decison

After defining the MDP tuple, we present our actor-critic network design. We describe its input, output and architecture.

In an advantage actor-critic model [13], the actor network, i.e., the policy network, learns a policy $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$, and produces actions. The critic network, i.e., the value network, estimates the state value $V(s)$.

To learn a joint caching policy using an actor network parameterized by θ, we can define the joint caching policy $\pi_\theta(\mathbf{a}|s)$ as follows,

$$\pi_\theta(\mathbf{a}|s) = \pi_\theta^1(a^1|s)\pi_\theta^2(a^2|s) = \prod_{a^i \in a^1} \pi_\theta^1(a^i|s) \prod_{a^j \in a^2} \pi_\theta^2(a^j|s). \tag{3}$$

In Sec. II-B we list different use cases, which involve two different action spaces: categorical action space, and discrete action space. Let us take Eq. (3) a further step by discussing these cases separately. Since cache size scaling is always a discrete control, we further denote it as $\pi_\theta^{1\mathcal{D}}(a^{1\mathcal{D}}|s)$ with $\mathcal{D}$ representing discrete action space. Though cache replacement strategy adaption has both categorical control case and discrete control case, it can be similarly further divided into two sub-sets: i) $\pi_\theta^{2\mathcal{C}}(a^{2\mathcal{C}}|s)$ with $\mathcal{C}$ representing categorical action space; ii) $\pi_\theta^{2\mathcal{D}}(a^{2\mathcal{D}}|s)$ with $\mathcal{D}$ representing discrete action space.

For categorical control, we represent the categorical policy $\pi^\mathcal{C}$ as a categorical distribution over K different choices parameterized by state-dependent probabilities:

$$\pi_\theta^\mathcal{C}(a^{2\mathcal{C}}|s) = Categorical(\alpha_{k,\theta}^i(s)), \quad \forall i, s \sum_{k=1}^{K} \alpha_{k,\theta}^i(s) = 1, \tag{4}$$

And for discrete control, taking $a^{1\mathcal{D}}$ as an example, we represent the discrete policy $\pi^\mathcal{D}$ as a discrete normal distribution which is derived as a discrete analog of the Gaussian distribution [14].

$$\begin{aligned} \pi_\theta^\mathcal{D}(a^{1\mathcal{D}}|s) = \mathcal{N}_{dis}(\mu_i(s), \sigma_i(s), \alpha_{k,\theta}^i(s)) \\ \forall i, s \sum_{k=1}^{K} \alpha_{k,\theta}^i(s) = 1, \end{aligned} \tag{5}$$

where θ is the parameters of an actor network that need to be optimized. In particular, $\alpha_\theta^i(s)$ represents as outputs of an actor network. $\mu_i(s)$ and $\sigma_i(s)$ are the mean and variance of discrete normal distribution that varies with different state s.

C. Discrepancy Between Discrete and Categorical Control Cases

In Fig. 4, we take examples to illustrate the differences between these two control cases. In the left column, the upper block shows a discrete case, where the content replacement strategy decision a^2 is δ_t. To achieve strategy adaptation, we can tune δ_t from $\{0.2, 0.4, 0.6, 0.8\}$. These four values are numerical variables and represented by four solid circles with different colors. The below block shows a categorical control case, where the strategy adaptation decision a^2 also has four values represented by four solid circles with different colors,

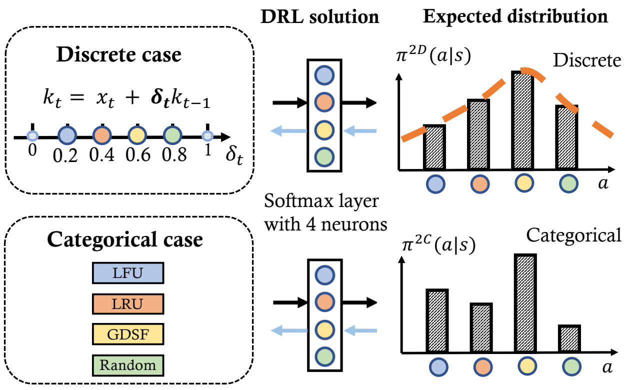
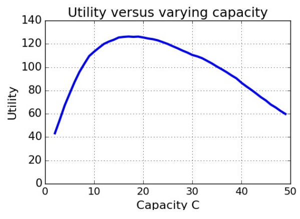
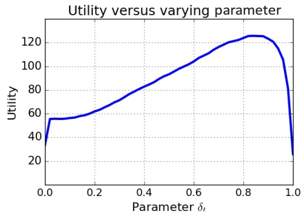


Fig. 4: Existing DRL algorithms treat both discrete scenarios (i.e., values) and categorical scenarios (i.e., different strategies) as categorical control and classification problem, resulting in performance degradation for discrete control cases.



(a) Varying cache size C



(b) Varying δ_t

Fig. 5: (a) Total utility with varying cache capacity. Taking the discrete control scenario as example, we fix $\delta_t \equiv 0.85$ and vary cache size C in range $[0, 50]$ with an interval 1. (b) Total utility with varying δ_t in range $[0, 1]$ with interval 0.02. Note that we fix cache size C as 20 for all $\{t|t = 0, 1, \cdots, T-1\}$.

corresponding to four different replacement strategies: {LFU, LRU, GDSF, Random}.

However, the discrete case poses a challenge to existing DRL algorithms, which tackle these two cases in the same way: setting the final layer as a softmax layer with four neurons, as shown in the middle column in Fig. 4. The softmax layer is used to output multi-class probabilities and inherently suitable for categorical control cases. Nevertheless, for discrete control cases, without considering the intrinsic order between discrete variables, simply using the softmax layer will cause performance degradation. As shown in the last column in Fig. 4, *the expected action distribution of the discrete case looks like a normal distribution, which may be inconsistent with the actual output of the softmax layer.*

It is obvious that the output action distributions of categorical control and discrete control are different. *However, existing DRL algorithms solve both of them as a classification problem and set the last layer of actor network as a softmax layer.* That is, we face a challenge that existing DRL algorithms treat them both as categorical control resulting in performance degradation for discrete control cases.

D. Distribution-Guided Regularizer

To reduce the performance degradation and design a guided distribution, we need to understand the characteristics of discrete control cases in our problem, e.g., is it unimodal or multimodal? Thus, we investigate how the utility varies with different cache sizes and different strategies, using the same trace with that of Sec. IV.

For the varying cache size C, at every timeslot t, we compute content score with $\delta_t \equiv 0.9$. Then we cache the top C contents with the highest content scores and compute the total utility. As shown in Fig. 5a, this total utility curve is unimodal: it increases first and then descends with cache size C. Note that the total utility does not increase monotonically due to rental costs.

For varying δ_t, we fix cache size $C \equiv 20$. At every timeslot t, we compute content score. Then we cache the top C contents with the highest content scores and compute the total utility. As shown in Fig. 5b, this curve is also unimodal.

Based on these observations, we add a discriminator for categorical control and discrete control cases. For discrete control, we design a unimodal distribution-guided policy $\hat{\mathcal{N}}_{dis}$, and use a Wasserstein distance loss to constrain the action probability distribution to be smoother, reducing performance degradation for discrete control cases. When updating from an old policy π to a new policy π', we denote our distribution-guided loss as $W(\pi')$,

$$W(\pi, \pi') = \sum_{i=1}^{2} x^i \phi(\pi^i, \pi'^i), \quad (6)$$

where,

$$\phi(\pi^i, \pi'^i) = |h(\pi'^i) - h(\pi^i)| D_W^{max}(\mathcal{N}_{dis}(h(\pi^i), \sigma), \pi'^i), \quad (7)$$

where x^i is an indicator variable that output by the discriminator: $x^i = 1$ if a^i belongs to discrete action space, and $x^i = 0$ if a^i belongs to categorical action space. $h(\pi^i) = \arg\max_{a^i} \pi^i(a^i|s)$ represents the maximum value of output action distribution generating by the old policy π on state s. $D_W^{max}(\mathcal{N}_{dis}(h(\pi^i), \sigma), \pi'^i)$ is the maximum Wasserstein distance of the guided distribution $\mathcal{N}_{dis}(h(\pi^i), \sigma)$ and the new policy π'^i over all states, which is $D_W^{max}(\cdot) = \max_s D_W(\mathcal{N}_{dis}(h(\pi^i), \sigma), \pi'^i(\cdot|s))$. Here σ is a constant.

E. Optimizing the Objective Function

Our objective function I_i to optimize is as follows:

$$\max_{\pi'} \mathbb{E}_{\tau\sim\pi'} \left[\sum_{t=0}^{T-1} \gamma^t A_\pi(s_t, \mathbf{a}_t) \right] - \zeta D_{KL}^{max}(\pi, \pi') - \beta W(\pi, \pi'), \quad (8)$$

where advantage $A_\pi(s_t, \mathbf{a}_t)$ means how much better it is to take a specific action $\mathbf{a}_t$ compared to the average action at the given state s_t. Advantage can be estimated by value network. Policy update loss $D_{KL}^{max}(\pi, \pi')$ is used to constrain on the size of the policy update, thus to avoid destructively large updates. $W(\pi, \pi')$ is a distribution-guided loss which is used to preserve intrinsic order of discrete variables. ζ and β are the coefficients of policy update loss and distribution-guided loss, respectively. A detailed analysis of the objective will be given in next subsection.

Next we will present our algorithm that maximizes our objective I_i in each update step on our joint caching policy. Specifically, given the old policy π, we select the new policy π' that maximizes I_i, leading to performance improvement.

According to importance sampling, when we need to compute an expected return of an unknown distribution p given distribution q, we can transform the expected return of distribution p as $E_{x\sim p}[f(x)] = E_{x\sim q}\left[f(x)\frac{p(x)}{q(x)}\right]$ with a constraint that distribution p and q should be close to each other, otherwise the estimation will have a large variance. In our problem, we can only sample data from distribution π, then the first term in Eq. (8) can be approximated as

$$\begin{aligned}
&\mathbb{E}_{\tau\sim\pi'}\left[A_\pi(s_t,\mathbf{a_t})\right]\\
=&\mathbb{E}_{\tau\sim\pi}\left[\frac{p_{\pi'}(s_t,\mathbf{a_t})}{p_\pi(s_t,\mathbf{a_t})}A_\pi(s_t,\mathbf{a_t})\right]\\
=&\mathbb{E}_{\tau\sim\pi}\left[\frac{\pi'(\mathbf{a_t}|s_t)}{\pi(\mathbf{a_t}|s_t)}\frac{\rho_{\pi'}(s_t)}{\rho_\pi(s_t)}A_\pi(s_t,\mathbf{a_t})\right]\\
=&\mathbb{E}_{\tau\sim\pi}\left[\frac{\pi'(\mathbf{a_t}|s_t)}{\pi(\mathbf{a_t}|s_t)}A_\pi(s_t,\mathbf{a_t})\right].
\end{aligned} \tag{9}$$

Note that the term $\frac{\rho_{\pi'}(s_t)}{\rho_\pi(s_t)}$ can be assumed to be 1 by ignoring changes in state visitation density [13]. Therefore, we can derive the following equivalent objective, which samples trajectory and data from old policy π to update new policy π':

$$\begin{aligned}
&\max_{\pi'}\ \mathbb{E}_{\tau\sim\pi}[L_{ori}-\beta W(\pi,\pi')].\\
&L_{ori}=\sum_{t=0}^{T-1}\gamma^t\frac{\pi'(\mathbf{a_t}|s_t)}{\pi(\mathbf{a_t}|s_t)}A_\pi(s_t,\mathbf{a_t})-\zeta D_{KL}^{max}(\pi,\pi')
\end{aligned} \tag{10}$$

F. Algorithm

We summarize JEANA in Algorithm 1.

```
Algorithm 1: JEANA
1  Randomly initialize critic network $V(\cdot)$ and actor
     network $\pi(\cdot)$ with parameters $\omega$ and $\theta$, respectively;
2  Initialize learning rates $\alpha_\theta$, $\alpha_\omega$;
3  Initialize batch $\mathbf{B}$;
4  Receive initial observed state $s_0$;
5  /**Decision Epoch**/
6  for $t$ = 0, $\cdots$, T-1 do
7      Sample $\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|s_t)$;
8      Execute action $\mathbf{a}_t$, observe reward $r_t$ and next state
         $s_{t+1} \sim P(s_{t+1}|s_t,\mathbf{a}_t)$;
9      Store transition sample $(s_t,\mathbf{a}_t,r_t,s_{t+1})$ into $\mathbf{B}$;
10 end
11 Sample trajectories from $\mathbf{B}$;
12 Compute advantage $A(s_t,a_t)$ based on current critic
     network $V_\omega$ using Eq. (12);
13 /**Distribution-Guided Loss**/
14 for $i$= 1, 2 do
15     Discriminate action spaces for joint action and set
         indicator $x^i$;
16     Compute $h(\pi^i)=\arg\max_{a^i}\pi_\theta^i(a^i|s)$;
17 end
18 Compute distribution-guided loss $W$ using Eq. (6);
19 /**Network Updating**/
20 Update $\theta$ by gradient ascent method w.r.t. Eq. (10);
21 Update $\omega$ by gradient descent method by regression on
     mean-squared error: $\omega =$
     $\arg\min_\omega \frac{1}{|\mathbf{B}|T}\sum_{\tau\in B}\sum_{t=0}^{T-1}(V_{\omega(s_t)}-\sum_{t'>t}\gamma^{t'-t}r_{t'})$.
```

G. Analysis on Performance Improvement Gurantee

After presenting the design of JEANA, we now provide analysis to prove that: i) optimizing our objective I_i is equivalent to maximizing long-term expected reward, i.e., utility of small content providers; ii) by building a lower bound of the expected reward, we can gurantee monotonic performance improvement by maximizing the lower bound.

To maximize the expected long-term reward η during a finite time horizon T, we define our goal as follows:

$$\eta(\pi)=E_{\tau\sim\pi}\left[\sum_{t=0}^{T-1}\gamma^t r_t\right], \tag{11}$$

The advantage can be estimated as

$$A_\pi(s,\mathbf{a})=\mathbb{E}_{s'\sim P(s'|s,\mathbf{a})}[r(s)+\gamma V_\pi(s')-V_\pi(s)], \tag{12}$$

Lemma 1: Expected long-term reward of a new policy π' can be expressed in terms of the advantage function over an old policy π.

$$\eta(\pi')=\eta(\pi)+\mathbb{E}_{\tau\sim\pi'}\left[\sum_{t=0}^{T-1}\gamma^t A_\pi(s_t,\mathbf{a}_t)\right]. \tag{13}$$

The proof can be found in [13]. And it can be rewritten as

$$\begin{aligned}
\eta(\pi')&=\eta(\pi)+\sum_{t=0}^{T-t}\sum_s p(s_t=s|\pi')\sum_{\mathbf{a}}\pi'(\mathbf{a}|s)\gamma^t A_\pi(s,\mathbf{a})\\
&=\eta(\pi)+\sum_s\sum_{t=0}^{T-t}\gamma^t p(s_t=s|\pi')\sum_{\mathbf{a}}\pi'(\mathbf{a}|s)A_\pi(s,\mathbf{a})\\
&=\eta(\pi)+\sum_s\rho_{\pi'}(s)\sum_{\mathbf{a}}\pi'(\mathbf{a}|s)A_\pi(s,\mathbf{a}),
\end{aligned} \tag{14}$$

where $\rho_\pi(s)$ is the discounted state visitation probability. $\eta(\pi')$ is not easy to be directly optimized since there exists $\rho_{\pi'}$, which contains π' that is unknown.

To solve this, we first make an approximation of $\eta(\pi')$. We substitute ρ_π for $\rho_{\pi'}$ by ignoring changes in state visitation density when policy updates from π to π', which yields,

$$L_\pi(\pi')=\eta(\pi)+\sum_s\rho_\pi(s)\sum_{\mathbf{a}}\pi'(\mathbf{a}|s)A_\pi(s,\mathbf{a}). \tag{15}$$

According to [15], a sufficiently small step $\pi \rightarrow \pi'$ that improves L_π will also improve η. Here comes a question: how to measure the policy update step? Thus, we need a distance measure between π and π'. We denote $D_{KL}^{max}(\pi||\pi')$ as the maximum Kullback-Leibler (KL) divergence of the old policy and the new policy over all states, which is expressed as $D_{KL}^{max}(\pi||\pi') = \max_s D_{KL}(\pi(\cdot|s)||\pi'(\cdot|s))$. And we denote $\epsilon = max_{s,\mathbf{a}}|A_\pi(s,\mathbf{a})|$. Given the notations above, we have the following theorem on a performance lower bound, the proof of which is presented in [13].

Theorem 1. *The expected performance of a new policy is bounded from below as follows:*

$$\begin{aligned}\eta(\pi') &\geq L_\pi(\pi') - \zeta D_{KL}^{max}(\pi, \pi'),\\ \text{where } \zeta &= \frac{4\epsilon\gamma}{(1-\gamma)^2}.\end{aligned} \tag{16}$$

From this theorem, we can derive the following corollary that proves the performance improvement guarantee.

Corollary 1. *A policy update can guarantee to generate a monotonically improving sequence of policies $\eta(\pi_0) \leq \eta(\pi_1) \leq \eta(\pi_2) \leq \cdots \leq \eta(\pi_i) \leq \eta(\pi_{i+1})$.*

Proof. Let $I_i(\pi) = M_i(\pi) - W_i(\pi)$, where $M_i(\pi) = L_{\pi_i}(\pi) - \zeta D_{KL}^{max}(\pi_i, \pi)$, and $W_i(\pi) = \sum_{i=1}^{2} x^i \phi(\pi^i, \pi'^i)$, then

$$\begin{aligned}&W_i(\pi_{i+1}) \geq 0,\\ &W_i(\pi_i) = 0,\\ &\eta(\pi_{i+1}) \geq M_i(\pi_{i+1}),\\ &\eta(\pi_i) = M_i(\pi_i).\\ &\eta(\pi_{i+1}) \geq M_i(\pi_{i+1}) - W_i(\pi_{i+1}) = I_i(\pi_{i+1}),\\ &\eta(\pi_i) = M_i(\pi_i) - W_i(\pi_i) = I_i(\pi_i),\\ &\eta(\pi_{i+1}) - \eta(\pi_i) \geq I_i(\pi_{i+1}) - I_i(\pi_i).\end{aligned} \tag{17}$$

By maximizing the lower bound I_i at each iteration, we guarantee the true objective performance η is non-decreasing. Therefore, I_i is an equivalent objective function.

IV. Evaluation

We conduct experiments on real-world video traces to evaluate JEANA's performances in different use cases, including categorical control and discrete control. We validate 1) the superiority of joint caching decision over strategy-only and size-only methods; 2) the stability of our JEANA training process; 3) the effectiveness of distribution-guided regularizer in reducing performance degradation.

A. Setup

Trace-Driven Simulation Environment. We first build a trace-driven environment to simulate the process of joint adaptive caching in elastic CDNs. We use a video request dataset, collected by a video content provider. Each request records the following attributes: i) the device id; ii) the timestamp; ii) the latitude and longitude coordinates of the current request; iv) the title of the video. This video request dataset covers $212,235$ requests to $33,125$ unique contents during 13 days of May, 2015. We aggregate the request data on an hourly basis to produce a content request trace. As shown in Sec. II-A, CP continuously receives requests. At each timeslot t, CP needs to determine which content replacement strategy to choose and how many cache units to rent from SP. We set the parameters of the SP's pricing function according to [16]. The SP charges CP for caching content j using a cache-time decreasing price function $p(v_t^j) = a\psi^{v_t^j} + b$, where $a = 0.017, \psi = 0.999888, b = 0.01$; v_t^j represents how many consecutive timeslots that content j has been cached. Thus, the rental cost is $d_t = \sum_{j=1}^{a_t^1} p(v_t^j)$. The initial leased cache unit cost for caching a new content is 0.027. The rationale behind this is that the longer a content is cached, the lower the unit cost it takes, which discourages frequent hard disk writing and data transmission.

We split the 13-day data into training and test parts with a ratio of 11:2. We build up a trace-driven training environment and a test environment separately.

For other parameters, we set $\lambda = 1.4676 \times 10^{-2}$ according to [3], and set $d = 48$. d means the length of past request information we consider. Note that we find d can range from 6 to 48, which does not affect the performance.

For categorical control case, $\mathcal{A} = [\mathcal{A}^{1\mathcal{D}}, \mathcal{A}^{2\mathcal{C}}]$, we set the dimension of action space as $[100, 3]$, which means cache size ranges from 0 to 99 with interval 1, and content replacement strategy shifts among LFU, LRU and $Random$. For discrete control case, $\mathcal{A} = [\mathcal{A}^{1\mathcal{D}}, \mathcal{A}^{2\mathcal{D}}]$, we set the dimension of action space as $[100, 101]$, which means cache size ranges from 0 to 99 with interval 1, and content replacement strategy parameter δ_t ranges from 0 to 1 with interval 0.01.

Actor-Critic Network. The actor network for learning a policy is a fully connected network (FCN) with four hidden layers, using ReLU [17] as the activation function. The critic network for value function estimation is a FCN with four hidden layers as well, but its final output is a linear neuron without activation function. The discount factor $\gamma = 0.99$. The initial learning rate is 1.25×10^{-5}, the minibatch number is 4, and the total timestep is 1.6×10^5. For distribution-guided regularizer, $\beta = 10^{-4}$, $\sigma = 1$. Our experiments run over 12 Intel Xeon 2.2GHz CPUs and 4 Nvidia GTX 1080Ti GPUs.

Baselines. We use the following baselines:

- **BestFixed**: best fixed decison in hindsight. It is obtained when we assume we know the future requests. $[a_{bs}^1, a_{bs}^2]$ represents the best fixed joint caching decision tuple, where a_{bs}^1 is the best value for cache size scaling, and a_{bs}^2 is the best value for replacement strategy.
- **JEANA-a^2**: JEANA without replacement strategy adaptation. It is a size-only method which adjusts a^1 only. It fixes content replacement strategy variable with a_{bs}^2.
- **JEANA-a^1**: JEANA without cache size scaling a^1. It is a strategy-only method which adjusts a^2 only. It fixes cache size scaling action with a_{bs}^1.
- **JEANA-W**: JEANA without distribution-guided loss. As shown in Eq. (10), the objective function of JEANA-W does not use the distribution loss $W(\pi, \pi')$.

Metrics. We use the following performance metrics:

- **Hit gain**: $G = \sum_{t=0}^{T-1} \lambda h_t$. The higher hit gain means the more requests served by elastic cache.
- **Rental cost**: $D = \sum_{t=0}^{T-1} d_t$. The more cache units we rent, the higher rental cost we spend.
- **Total reward**: $R = G - D$. The higher total reward means a better adaptive cache rental decision.

B. Ablation Study

To understand the impact of each component in JEANA, we conduct ablation studies to demonstrate: i) the superiority of joint caching decision; ii) distribution-guided loss contributes to the overall performance; iii) performance improvement guarantee stabilizes the training process.

*1) **Superiority of joint caching decision**:* Table II shows the metric statistics of JEANA and several baselines. Our experimental results confirm some conclusions in Sec. I: 1) Size-only adaptive caching methods have a relatively low hit rate without strategy switching cost, therefore resulting in low cache rental cost. 2) Strategy-only adaptive caching methods have a high cache hit rate with high strategy switching cost, therefore resulting in a high cache rental cost. Joint adaptive caching can get the best overall performance. As shown in Table II, size-only method JEANA-a^2 achieves the lowest hit gain no matter in Cat. or Dis. cases. JEANA always achieves the highest hit gain, which is 25% and 23.3% higher than size-only method JEANA-a^2, and 3.5% and 7.3% higher than strategy-only method JEANA-a^1. The superiority of joint caching over strategy-only comes from lower rental cost, with 3.83% and 1.0% lesser rental cost.

As for the total reward, in categorical control case and discrete control case, JEANA outperforms BestFixed by 23.14% and 22.73%, respectively; outperforms single cache size scaling baseline JEANA-a^2 by 13.57% and 13.26%, respectively; outperforms single content replacement strategy adaptation baseline JEANA-a^1 by 16.84% and 21.53%, respectively.

TABLE II: Performance metrics of different methods for categorical case (Cat.) and discrete case (Dis.).

Methods	Hit gain		Rental cost		Total reward	
	Cat.	Dis.	Cat.	Dis.	Cat.	Dis.
BestFixed	27.65	30.15	19.01	19.02	8.64	11.13
JEANA-a^2	22.56	27.24	13.28	15.18	9.28	12.06
JEANA-a^1	27.06	31.26	17.92	20.02	9.14	11.24
JEANA-W	27.67	33.3	17.29	20.08	10.38	13.22
JEANA	**28.00**	**33.55**	**17.26**	**19.89**	**10.78**	**13.66**

From the action sequence in the categorical control scenario as shown in Fig. 7, the strategy-only method JEANA-a^1 switches the strategy a total of 7 times, while JEANA only switches the strategy 3 times in total. Size-only method JEANA-a^1 scales the size only 5 times while JEANA scales size 11 times in total. JEANA achieves the best total reward performance while saving the cost of strategy switching and get the highest hit rate. Besides, there is time consistency between strategy switching and size scaling.

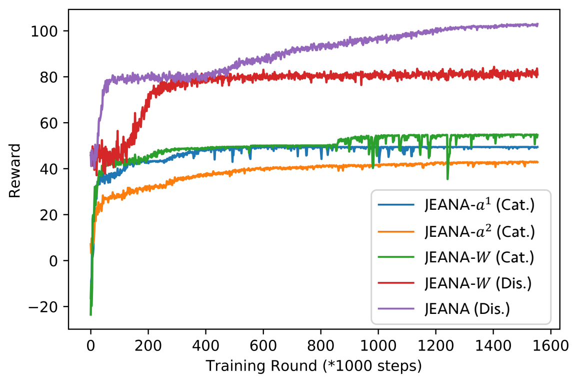


Fig. 6: Training process and episode reward of JEANA-a^1, JEANA-a^2, JEANA-W and JEANA.

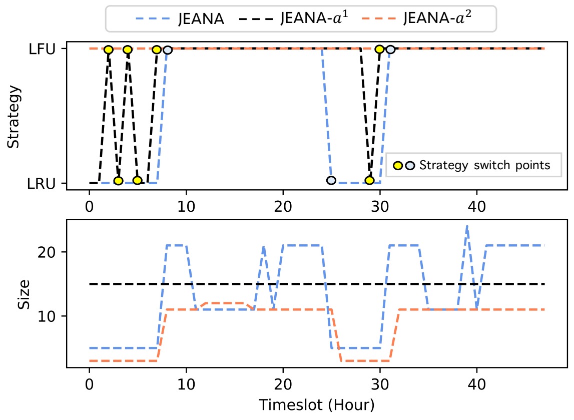


Fig. 7: Actions generated by JEANA, JEANA-a^2 and JEANA-a^1 on 2-day test data when $\mathcal{A} = [\mathcal{A}^{1\mathcal{D}}, \mathcal{A}^{2\mathcal{C}}]$.

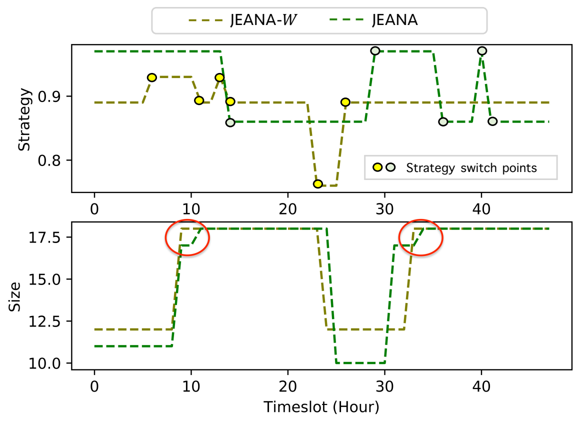


Fig. 8: Actions generated by JEANA-W and JEANA on 2-day test data when $\mathcal{A} = [\mathcal{A}^{1\mathcal{D}}, \mathcal{A}^{2\mathcal{D}}]$.

*2) **Distribution-guided loss contributes to performance**:* we evaluate the performance improvement contributed by distribution-guided loss W. Therefore, we compare JEANA over baseline JEANA-W.

The performance comparison of them is given in the last two rows in Table II. JEANA outperforms JEANA-W in both cases on the total reward metric, while the performance improvement in categorical control case (2.50%) is smaller than that in the discrete control case (3.32%). The reason is that, for categorical control, only cache size scaling action a^1 is discrete value, while for discrete control case, both a^1 and a^2 are discrete values, which results in a greater improvement.

From the action sequence in the discrete control scenario as shown in Fig. 8, JEANA-W switched strategies for a total of 6 times, while JEANA only switched strategies for 5 times. JEANA achieved the best total reward performance while saving the cost of strategy switching. Besides, there is time consistency between the strategy switching and size scaling.

3) ***Performance improvement guarantee stabilizes the training process:*** we run experiments to prove that the performance improvement guarantee of JEANA stabilizes the training process of learning a joint caching policy.

We iteratively train our JEANA model on the training data. Fig. 6 represents the training process and episode reward of JEANA-a^1, JEANA-a^2, JEANA-W, and JEANA. JEANA achieves higher episode reward than other baselines. The episode reward of JEANA continuously increases and finally converges. Besides, comparing to JEANA-W, JEANA performs more stable, especially at the latter phase of training.

V. Related Work

A. Adaptive Caching

Size-only strategies focuses on adaptive cache resource allocation or adaptive cache storage rental. Dehghan *et al.* [4] proposed a utility-driven cache partitioning approach for multiple content providers. Chu *et al.* [18] formulated the problem of jointly optimizing cache resource allocation and request routing as mixed-integer programming. These works made assumptions on inter-request time distributions or request rate variations. Kwak *et al.* [3] first considered the problem of dynamic cache rental and content caching in elastic wireless CDNs. It selects investment, placement, and request association to maximize the utility of content providers. Carra *et al.* [19] considered elastic cloud caching. They proposed an adaptive TTL-based solution to dynamically track the required cache size to minimize the total cost, which contains the sum of storage cost and cost due to the misses.

Strategy-only strategies adaptively adjust the replacement strategy to improve performance. LRFU [6] provided a spectrum of policies using a parameterized function to determine the effects of recency and frequency factors on the likelihood of future re-reference. Adaptive LRFU (ALRFU), dynamically adjusts the parameter [7]. An online self-tuning method called adaptive replacement cache (ARC) were developed [8], it adapted cache partitions while had constant complexity.

B. Deep Reinforcement Learning

DRL can be roughly divided into four categories, depending on whether the action space handled is continuous or discrete. The first category includes Deep Q-network (DQN) [20] and its variants such as Double Q-learning [21] can only handle discrete action spaces. Deep Deterministic Policy Gradients (DDPG) [22] and its variant Twin Delayed DDPG (TD3) are model-free RL algorithms designed for handling continuous action spaces. Soft actor-critic (SAC) [23] is a DRL framework for training maximum entropy policies in continuous action spaces. Asynchronous Advantage Actor-Critic (A3C) [24] and its single-worker version Advantage Actor-Critic (A2C) [25] are capable of handling both discrete and continuous action spaces. Trust Region Policy Optimization (TRPO) [13] and Proximal Policy Optimization (PPO) [26] can also deal with both discrete and continuous action spaces.

C. RL for Caching

Recently, RL methods have applied in caching problems. They cast the caching problem as an MDP problem and then solve it using RL methods. [10] used the Q-learning approach to derive the optimal fetching-caching strategy to minimize the overall cost of expenditure. However, Q-learning is only applicable to scenarios with low-dimensional state-action space, and it is difficult to apply to problems with large state-action spaces. In [9], the proactive caching problem in the dynamic content request scenario is modeled as an MDP, and the optimal threshold in the caching strategy is obtained by the strategy gradient method to minimize the long-term average energy cost. To solve the content cache of the base station at the edge node, [11] added the Wolpertinger method to the DDPG framework to maximize the long-term cache hit rate. [27] proposed RL-cache, which models the cache admission problem in the CDN cache as an RL problem and solves it with the Monte Carlo method to maximize the cache hit ratio. Besides, the authors selected the content size, request frequency, and most recently requested features as features to train the model.

VI. Conclusion

To simultaneously optimize two different variables, *i.e.,* categorical and discrete variable in joint elastic caching problem, we propose a distribution-guided reinforcement learning framework JEANA to learn the joint policy. It utilizes a distribution-guided regularizer based on Wasserstein distance to constrain the action distribution to be smoother and capture continuity among discrete variables, which improves model representation and capacity. Furthermore, we prove that this algorithm has a theoretical guarantee of monotonic policy improvement. We demonstrate the effectiveness of our method on real video-on-demand request traces, which shows JEANA improves the cache hit ratio while reducing the rental cost.

VII. Acknowledgement

This work is supported in part by the National Key Research and Development Program of China under No. 2020AAA0106300, NSFC under Grant No. 61872215, No. 61971457 and No. 62050110, SZSTI under Grant No. JCYJ20180306174057899, and Shenzhen Nanshan District Ling-Hang Team Project under Grant No. LHTD20170005.

References

[1] "Akamai collaborates with orange on nfv initiative to dynamically scale cdn capacity for large event," in *[Online]. Available: https://www.akamai.com/us/en/about/news/press/2016-press/akamai-collaborates-with-orange-on-nfv-initiative.jsp*.

[2] Huawei, "Huawei ucdn solution," in *[Online]. Available: http://carrier.huawei.com/en/solutions/cloud-powered-digital-services/ucdn*.

[3] J. Kwak, G. Paschos, and G. Iosifidis, "Dynamic cache rental and content caching in elastic wireless cdns," in *2018 16th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*. IEEE, 2018, pp. 1–8.

[4] M. Dehghan, W. Chu, P. Nain, and D. Towsley, "Sharing LRU Cache Resources among Content Providers : A Utility-Based Approach," *IEEE/ACM Transactions on Networking*, vol. 27, no. 2, pp. 477–490, 2019.

[5] W. Chu, M. Dehghan, D. Towsley, and Z.-l. Zhang, "On Allocating Cache Resources to Content Providers," in *Proceedings of the 3rd ACM Conference on Information-Centric Networking*, 2016, pp. 154–159.

[6] D. Lee, J. Choi, J.-H. Kim, S. H. Noh, S. L. Min, Y. Cho, and C. S. Kim, "Lrfu: A spectrum of policies that subsumes the least recently used and least frequently used policies," *IEEE transactions on Computers*, no. 12, pp. 1352–1361, 2001.

[7] D. Lee, J. Choi, J.-H. Kim, S. H. Noh, S. L. Min, Y. Cho, and C.-S. Kim, "On the existence of a spectrum of policies that subsumes the least recently used (lru) and least frequently used (lfu) policies." in *SIGMETRICS*, vol. 99. Citeseer, 1999, pp. 1–4.

[8] N. Megiddo and D. S. Modha, "Arc: A self-tuning, low overhead replacement cache." in *FAST*, vol. 3, no. 2003, 2003, pp. 115–130.

[9] S. O. Somuyiwa, A. György, and D. Gündüz, "A reinforcement-learning approach to proactive caching in wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 6, pp. 1331–1344, 2018.

[10] 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*, 2019.

[11] C. Zhong, M. C. Gursoy, and S. 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.

[12] S. Basu, A. Sundarrajan, J. Ghaderi, S. Shakkottai, and R. Sitaraman, "Adaptive ttl-based caching for content delivery," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 45, no. 1. ACM, 2017, pp. 45–46.

[13] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," in *International conference on machine learning*, 2015, pp. 1889–1897.

[14] A. W. Kemp, "Characterizations of a discrete normal distribution," *Journal of Statistical Planning and Inference*, vol. 63, no. 2, pp. 223–229, 1997.

[15] S. Kakade and J. Langford, "Approximately optimal approximate reinforcement learning," in *ICML*, vol. 2, 2002, pp. 267–274.

[16] "Amazon elastic cdn service - elasticache," in *[Online]. Available: https://aws.amazon.com/elasticache/*.

[17] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *ICML*, 2010.

[18] W. Chu, M. Dehghan, J. C. S. Lui, D. Towsley, and Z.-l. Zhang, "Joint Cache Resource Allocation and Request Routing for In-network Caching Services," *Computer Networks*, vol. 131, pp. 1–14, 2018.

[19] D. Carra, G. Neglia, and P. Michiardi, "TTL-based Cloud Caches," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 685–693.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[21] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *AAAI'16 Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2016, pp. 2094–2100.

[22] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," in *ICLR 2016 : International Conference on Learning Representations 2016*, 2016.

[23] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *ICLR 2018 : International Conference on Learning Representations 2018*, 2018.

[24] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *ICML'16 Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, 2016, pp. 1928–1937.

[25] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz, "Reinforcement learning through asynchronous advantage actor-critic on a gpu," *arXiv preprint arXiv:1611.06256*, 2016.

[26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms." *arXiv preprint arXiv:1707.06347*, 2017.

[27] V. Kirilin, A. Sundarrajan, S. Gorinsky, and R. K. Sitaraman, "RL-Cache : Learning-Based Cache Admission for Content Delivery," in *Proceedings of the 2019 Workshop on Network Meets AI & ML*, 2019, pp. 57–63.