

Cache-Aware Reinforcement Learning in Large-Scale Recommender Systems

Xiaoshuang Chen*
Kuaishou Technology
Beijing, China
chenxiaoshuang@kuaishou.com

Gengrui Zhang*
Kuaishou Technology
Beijing, China
zhanggengrui@kuaishou.com

Yao Wang
Kuaishou Technology
Beijing, China
wangyiyian@kuaishou.com

Yulin Wu
Kuaishou Technology
Beijing, China
wuyulin@kuaishou.com

Shuo Su
Kuaishou Technology
Beijing, China
sushuo@kuaishou.com

Kaiqiao Zhan
Kuaishou Technology
Beijing, China
zhankaiqiao@kuaishou.com

Ben Wang[†]
Kuaishou Technology
Beijing, China
wangben@kuaishou.com

ABSTRACT

Modern large-scale recommender systems are built upon computation-intensive infrastructure and usually suffer from a huge difference in traffic between peak and off-peak periods. In peak periods, it is challenging to perform real-time computation for each request due to the limited budget of computational resources. The recommendation with a cache is a solution to this problem, where a user-wise result cache is used to provide recommendations when the recommender system cannot afford a real-time computation. However, the cached recommendations are usually suboptimal compared to real-time computation, and it is challenging to determine the items in the cache for each user. In this paper, we provide a cache-aware reinforcement learning (CARL) method to jointly optimize the recommendation by real-time computation and by the cache. We formulate the problem as a Markov decision process with user states and a cache state, where the cache state represents whether the recommender system performs recommendations by real-time computation or by the cache. The computational load of the recommender system determines the cache state. We perform reinforcement learning based on such a model to improve user engagement over multiple requests. Moreover, we show that the cache will introduce a challenge called critic dependency, which deteriorates the performance of reinforcement learning. To tackle this challenge, we propose an eigenfunction learning (EL) method to learn independent critics for CARL. Experiments show that CARL

can significantly improve the users' engagement when considering the result cache. CARL has been fully launched in Kwai app, serving over 100 million users.

CCS CONCEPTS

• Information systems → Recommender systems.

KEYWORDS

Reinforcement Learning, Recommender Systems, Cache, Eigenfunction

ACM Reference Format:

Xiaoshuang Chen, Gengrui Zhang, Yao Wang, Yulin Wu, Shuo Su, Kaiqiao Zhan, and Ben Wang. 2024. Cache-Aware Reinforcement Learning in Large-Scale Recommender Systems. In *Companion Proceedings of the ACM Web Conference 2024 (WWW '24 Companion)*, May 13–17, 2024, Singapore, Singapore. ACM, New York, NY, USA, 8 pages. <https://doi.org/10.1145/3589335.3648326>

1 INTRODUCTION

Recently, reinforcement learning (RL) has drawn a growing attraction in recommender systems [1, 2, 4, 13, 15, 16]. RL-based recommendation methods treat users as the environment and the recommender system as the agent and then model the sequential interactions between users and the recommender system. RL achieves success in improving users' long-term engagements, such as the total dwell time [2, 15] and the user retention [1].

Existing RL methods require the system to interact with the users and change the next recommendation according to the user feedback in real-time. However, modern large-scale recommender systems are built upon computation-intensive infrastructure [12]. Although there is an amount of research on reducing the computational burden of recommender systems [6, 7, 9], it is still challenging to perform real-time computation for each request during peak periods. Fig. 1 shows the queries-per-second (QPS) of Kwai app based on the average QPS per day. It is shown that the computational burden during peak periods is several times that of off-peak periods. It

*Both authors contributed equally to this research.

[†]Corresponding author.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

WWW '24 Companion, May 13–17, 2024, Singapore, Singapore

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-0172-6/24/05...\$15.00

<https://doi.org/10.1145/3589335.3648326>

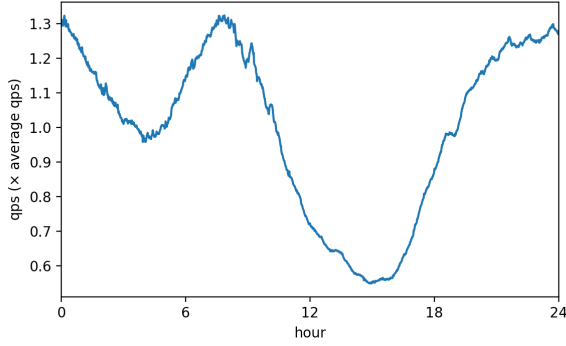


Figure 1: QPS in one day of Kwai app.

requires a lot of computational resources to perform real-time computation in the peak period, while the same resources will be idle in the off-peak periods, which is not cost-effective. Therefore, a result cache [5, 11] is used to balance the computational burden and the recommendation performance in large-scale recommender systems. Fig. 2 briefly shows the recommender system with a result cache in Kwai. When a user sends a request, the system first provides a real-time recommendation, which returns a group of L items, of which the top K items are showed to the user while the other $L - K$ are put into a result cache. When the user's next request comes, the recommender system will perform a real-time computation if the traffic does not exceed the system's affordability, e.g. Response 2 in Fig. 2. Otherwise, it will recommend K items directly from the result cache, e.g. Response 3 in Fig. 2. The existence of the cache mitigates the computational burden of the recommender systems in peak periods, but it brings several challenges to traditional RL approaches:

- **Lack of actions in the cache.** The RL module is usually a part of the real-time computation, meaning RL will not be performed when the request is processed by the cache. This contradicts the assumption of RL to interact with the users continuously.
- **Unpredictability of the cache choice.** The choice of using real-time computation or the cache depends on the total computational burden of the system rather than the features of the specific request. Therefore, the cache choice is unpredictable by the RL algorithm, which increases the learning difficulty.
- **Heterogeneous rewards.** the performance of cached results is suboptimal compared to the results from the real-time computation due to the lack of instant behaviors of the user. Table 1 provides an example in the Kwai app, where the average user engagement of cached recommendations is lower than that of real-time recommendations. Such characteristics need to be considered in the RL model.

To this end, we provide a novel cache-aware reinforcement learning (CARL) method. CARL introduces a cache state to model the choice of recommendations by real-time computation or by the cache. The cache state is determined by the computational burden of the recommender system and is independent of the user states

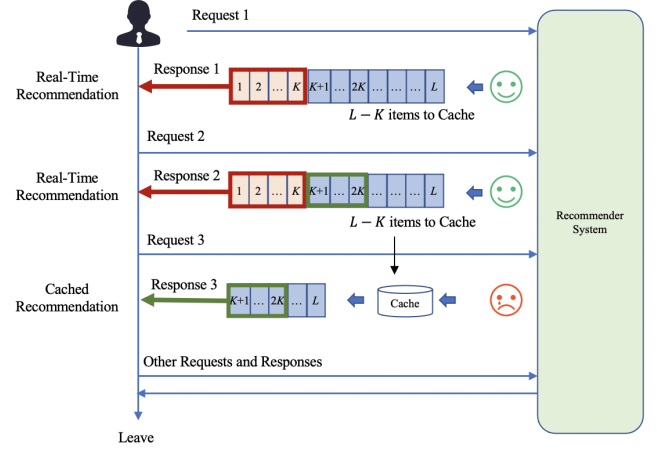


Figure 2: Recommendation with a result cache.

	Real-Time Recommendations	Cached Recommendations
Watch Time	$\times 1$	$\times 0.85$
Like Rate	$\times 1$	$\times 0.68$
Follow Rate	$\times 1$	$\times 0.54$

Table 1: Comparisons between User engagement in real-time and cached recommendations in Kwai.

and actions of the system. We also show that the existence of the cache introduces challenges to CARL training. Specifically, the uncontrollable cache state introduces an uncontrollable dependency between the critic functions of real-time cases and cached cases, which deteriorates the performance of the training algorithm. To tackle this challenge, we introduce a novel eigenfunction learning (EL) technique to train CARL. EL learns two independent critics and then combines them to obtain the critic functions of real-time and cached recommendations, improving CARL's performance.

In summary, our contributions are as follows:

- We introduce a novel CARL method to model the recommender systems with cache explicitly.
- We show that the existence of the cache introduces critic dependency to the RL algorithm, which deteriorates the performance. Then, we provide an EL algorithm to train CARL effectively.
- Experiments show that CARL improves users' engagement in recommender systems with the cache. CARL has been launched in Kwai app, serving over 100 millions of users.

Following the introduction, Section 2 provides the CARL model to describe the recommendation process with a cache. Section 3 discusses the challenges of learning CARL, and provides an effective EL algorithm to train CARL. Section 4 provides the experimental results, and Section 5 concludes the paper.

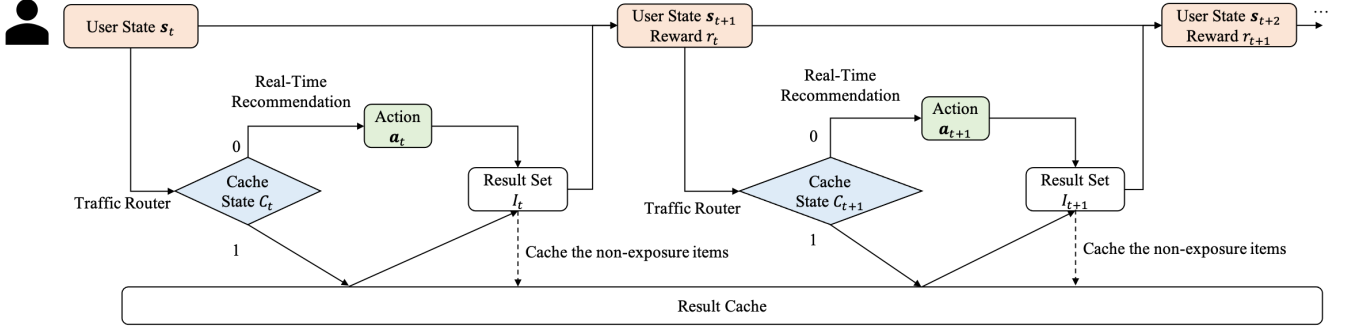


Figure 3: The CARL model.

2 MODELING OF CARL

This section provides the CARL model, which is an RL model considering the existence of the result cache, as shown in Figure 3. We model the interaction between users and the recommender systems as a Markov Decision Process (MDP), where the recommender system is the agent, and users are the environments. When a user opens the app, a session begins, which consists of multiple requests until the user leaves the app. At Step t , the recommender system obtains a user state s_t . The user state s_t consists of the user profile, behavior history, request context, and candidate video features. According to the user request with the state s_t , the recommender system generates an action a_t , and outputs a set of items I_t to the user according to the user state s_t and the action a_t . After watching the provided items in I_t , the user provides feedback r_t , e.g. the watching time of the items and other interactions on the items. Then, the user transfers to the next state s_{t+1} and determines whether to send the next request or leave. The recommender system aims to maximize the long-term reward R_t defined by

$$R_t = \sum_{t'=t}^T \gamma^{t'-t} \mathbb{E}[r_{t'}], \quad (1)$$

where T is the last step and γ is the discount factor.

A key component to be modeled in the recommender systems with cache is the choice of real-time computation or cached results. Specifically, the system maintains a result cache $\mathcal{K}_u$ for each user u . When a user request comes at Step t , a traffic router will provide a cache state $C_t \in \{0, 1\}$, and the system will provide different services according to C_t :

- **Recommendation by real-time computation.** When the recommender system has enough computational resources, the traffic router returns $C_t = 0$, and the recommender system will use a real-time computation to provide recommended items. Specifically, n deep models will be used to predict scores $\mathbf{x}_{j,t} \in \mathbb{R}^n$ for each candidate item j , in terms of various feedback (watch time, follow, like, etc.). Then, the action a_t is used to generate the final score of each item according to a parametrized ranking function f , i.e.

$$z_{j,t} = f(\mathbf{x}_{j,t}; \mathbf{a}_t) \quad (2)$$

Finally, the top L items regarding the scores $z_{j,t}$ are used, of which the top K items are recommended to the user, while

the items with rank $K + 1$ to L are put into the result cache $\mathcal{K}_u$ for future use.

- **Recommendation by the cache.** When the traffic of the recommender system is too large to afford a real-time computation, the traffic router returns $C_t = 1$, and the system will provide recommendations by the cache. In such cases, the recommender system obtains K items directly from the user's result cache $\mathcal{K}_u$. Then, $\mathcal{K}_u$ is updated by removing the K items recommended. There is no action a_t in recommendations by the cache.

In our scenario, the traffic router maintains a queue of ongoing real-time recommendation processes. When a new request comes, the traffic router checks the queue length. If the queue length does not exceed a given limit, it suggests a real-time computation; otherwise, it suggests a cached recommendation.

Given the abovementioned models, CARL aims to maximize the aggregate reward R_t defined in Eq. (1). The main challenge of training CARL arises from the stochastic characteristic of the traffic router. The cache state C_t depends on the total traffic of the system rather than the user state s_t . Therefore, C_t is unpredictable according to s_t . Such stochastic characteristics make it challenging to learn the critic. Section 3 will discuss the challenge in detail and provide an effective training method.

3 LEARNING OF CARL

This section first shows the application of traditional RL algorithms to CARL, and discusses the critic dependency problem of the direct learning approach. Then, we provide the EL algorithm to learn CARL effectively.

3.1 Challenges of Direct Learning

3.1.1 The Direct Learning Algorithm. We first show a direct application of traditional RL to learn the CARL model. We consider a typical actor-critic architecture. The action a_t is determined by a policy function μ with the parameter θ :

$$\mathbf{a}_t = \mu(s_t; \theta) \quad (3)$$

We use a critic function $Q(s_t, \mathbf{a}_t; \phi)$, parameterized by ϕ to estimate the long-term reward R_t given the state s_t and the action $\mathbf{a}_t$. A typical critic learning algorithm [8] uses a temporal difference

	$C_t = 0$	$C_t = 1$
$C_{t+1} = 0$	$\left[Q_0(s_t, \mathbf{a}_t; \phi_0) - \left(r_t + \gamma Q_0(s_{t+1}, \mu(s_{t+1}; \theta^-); \phi_0^-) \right) \right]^2$	$\left[Q_1(s_t; \phi_1) - \left(r_t + \gamma Q_0(s_{t+1}, \mu(s_{t+1}; \theta^-); \phi_0^-) \right) \right]^2$
$C_{t+1} = 1$	$\left[Q_0(s_t, \mathbf{a}_t; \phi_0) - \left(r_t + \gamma Q_1(s_{t+1}; \phi_1^-) \right) \right]^2$	$\left[Q_1(s_t, \mathbf{a}_t; \phi_1) - \left(r_t + \gamma Q_1(s_{t+1}; \phi_1^-) \right) \right]^2$

Table 2: Difference formulations of $l_{DL}(\phi)$ in Eq. (8)

method to learn the parameter ϕ of the critic Q :

$$l(\phi) = [Q(s_t, \mathbf{a}_t; \phi) - (r_t + \gamma Q(s_{t+1}, \mu(s_{t+1}; \theta^-); \phi^-))]^2 \quad (4)$$

where θ^- is the parameter of the target actor, and ϕ^- is the parameter of the target critic. The critic function Q is used as the reward function of the policy function μ , and the parameter θ of the actor updates according to the following gradient ascent:

$$\nabla_{\theta} J = \nabla_{\theta} \mu(s_t; \theta) \nabla_{\mu} Q(s_t, \mu(s_t; \theta)) \quad (5)$$

To apply Eq. (4)(5) to CARL, we define the conditional critic functions:

$$Q_0(s_t, \mathbf{a}_t; \phi_0) \triangleq \mathbb{E}[R_t | C_t = 0], Q_1(s_t; \phi_1) \triangleq \mathbb{E}[R_t | C_t = 1] \quad (6)$$

Q_0 and Q_1 are the expected long-term rewards under the condition that the Request t is processed by real-time computation and by the cache, respectively. The input of Q_1 does not contain the action $\mathbf{a}_t$ because there is no action in recommendations by the cache.

Given Q_0 and Q_1 , the total critic function Q can be written as

$$Q(s_t, \mathbf{a}_t; \phi) = Q_{C_t}(s_t, \mathbf{a}_t; \phi_{C_t}), C_t \in \{0, 1\} \quad (7)$$

where $\phi = \{\phi_0, \phi_1\}$. With the critic defined in Eq. (7), we can directly train the CARL by the following critic loss:

CARL-DL (Direct Learning of CARL):

$$l_{DL}(\phi) = [Q_{C_t}(s_t, \mathbf{a}_t; \phi) - (r_t + \gamma Q_{C_{t+1}}(s_{t+1}, \mu(s_{t+1}; \theta^-); \phi^-))]^2, \quad (8)$$

with the policy loss remains the same as Eq. (5).

3.1.2 Critic Dependency. The CARL-DL approach is straightforward, but it faces a considerable challenge, i.e. the **Critic Dependency**. Specifically, Eq. (8) shows the dependency of $Q(s_t, \mathbf{a}_t; \phi)$ and $Q(s_{t+1}, \mathbf{a}_{t+1}; \phi)$. According to different cache states C_t and C_{t+1} , there are four possible formulations of Eq. (8), as shown in Table 2. It is clear that the Q_0 and Q_1 depend on each other. Figure 4 also shows the dependency among the two critics Q_0 and Q_1 in the DL of CARL. The critic dependency makes the learning process depending on the values of the cache states C_t and C_{t+1} , but the cache states are highly stochastic, and are unpredictable by the states s_t and the actions $\mathbf{a}_t$. Such problem deteriorates the convergence of the critic learning.

To discuss this problem more precisely, we consider the transit function of the critics. Since the cache state C_t is a stochastic variable, we use $D_c(t)$ to denote the probability that $C_t = c$:

$$D_c(t) \triangleq P(C_t = c), c \in \{0, 1\} \quad (9)$$

$D_c(t)$ does not depend on the user's state s_t because it simply relies on the total requests per second and the maximal real-time recommendations per second.

In addition, we define the expectation of the immediate reward r_t of different cache states C_t :

$$V_0(s_t, \mathbf{a}_t) \triangleq \mathbb{E}[r_t | C_t = 0], V_1(s_t) \triangleq \mathbb{E}[r_t | C_t = 1] \quad (10)$$

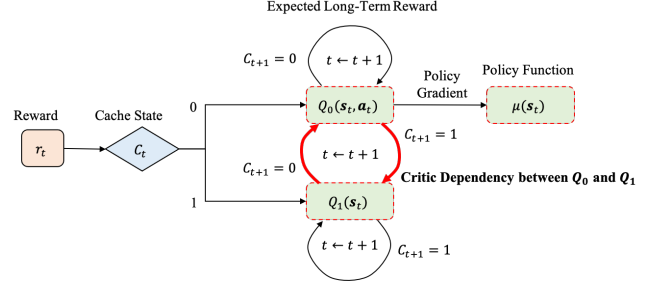


Figure 4: Direct Learning of CARL

Then, according to the backward induction, the transit functions of the expected immediate reward and the expected long-term reward can be written as:

$$\begin{aligned} Q_0(s_t, \mathbf{a}_t) &= V_0(s_t, \mathbf{a}_t) + \gamma D_0(t+1) \mathbb{E}_{\mathbf{a}_{t+1}} Q_0(s_{t+1}, \mathbf{a}_{t+1}) \\ &\quad + \gamma D_1(t+1) \mathbb{E}_{\mathbf{a}_{t+1}} Q_1(s_{t+1}) \\ Q_1(s_t) &= V_1(s_t) + \gamma D_0(t+1) \mathbb{E}_{\mathbf{a}_{t+1}} Q_0(s_{t+1}, \mathbf{a}_{t+1}) \\ &\quad + \gamma D_1(t+1) \mathbb{E}_{\mathbf{a}_{t+1}} Q_1(s_{t+1}) \end{aligned} \quad (11)$$

Now, we provide more explanations on Eq. (11), as shown in Figure 4. On the one hand, the critic functions Q_0 and Q_1 , i.e. the expectation of the long-term reward R_t , depends on the immediate reward r_t , estimated by V_0 and V_1 . On the other hand, Q_0 and Q_1 also depend on the future feedback of the user. However, due to the stochastic characteristic of the cache state C_t , the future feedback of the user obeys different distributions given different C_t , and the expectation of the future feedback is described by the second and third term of the right part of Eq. (11).

Eq. (11) also shows the critic dependency problem, since the Q_0 and Q_1 clearly depend on each other. Such dependency, together with the stochasticity of the cache states C_t , will increase the learning difficulty of CARL. We will discuss the solution to the critic dependency problem in the following subsections.

3.2 Eigenfunctions

This paper uses the eigenfunction technique to solve the critic dependency problem. The motivation is to find a group of independent variables derived from the critic functions Q_0 and Q_1 so that the learning process of different critic functions can be decoupled.

We first define the eigen immediate reward and the eigen long-term reward.

Definition 3.1 (Eigen Immediate Reward). The eigen immediate rewards, denoted by Γ_a and Γ_b , are given by

$$\begin{aligned} \Gamma_a(s_t, \mathbf{a}_t) &\triangleq V_0(s_t, \mathbf{a}_t) - V_1(s_t) \\ \Gamma_b(s_t, \mathbf{a}_t) &\triangleq D_0(t)V_0(s_t, \mathbf{a}_t) + D_1(t)V_1(s_t) \end{aligned} \quad (12)$$

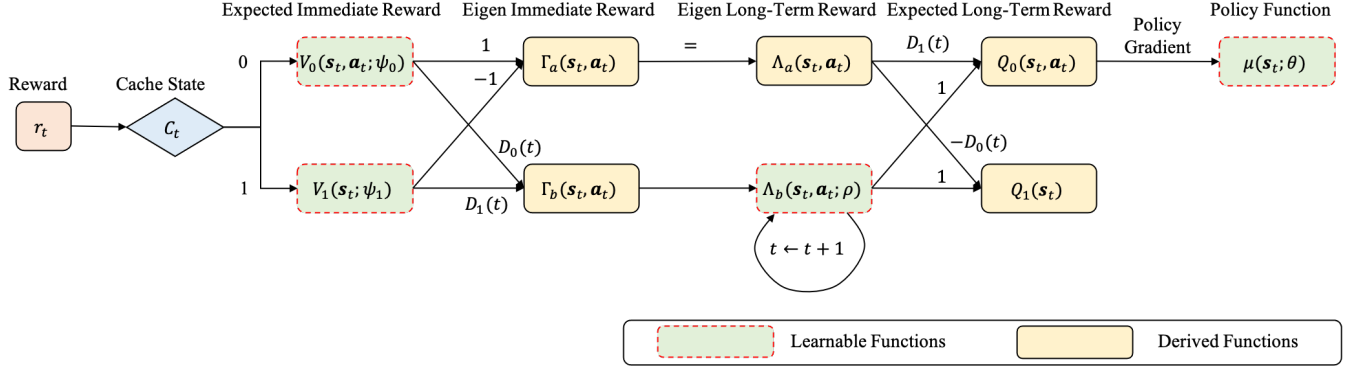


Figure 5: Eigenfunction Learning of CARL

The eigen immediate rewards Γ_a and Γ_b are the linear combinations of the expected immediate rewards V_0 and V_1 . Similarly, we have

Definition 3.2 (Eigen Long-Term Reward). The eigen long-term rewards, denoted by Λ_a and Λ_b , are given by

$$\begin{aligned}\Lambda_a(s_t, a_t) &\triangleq Q_0(s_t, a_t) - Q_1(s_t) \\ \Lambda_b(s_t, a_t) &\triangleq D_0(t)Q_0(s_t, a_t) + D_1(t)Q_1(s_t)\end{aligned}\quad (13)$$

Clearly, given the eigen long-term rewards Λ_a and Λ_b , we can recover the expected long-term rewards Q_0 and Q_1 simply by solve the equations in Eq. (13). Specifically, we have

$$\begin{aligned}Q_0(s_t, a_t) &= D_1(t)\Lambda_a(s_t, a_t) + \Lambda_b(s_t, a_t) \\ Q_1(s_t) &= -D_0(t)\Lambda_a(s_t, a_t) + \Lambda_b(s_t, a_t)\end{aligned}\quad (14)$$

Therefore, once we can estimate the eigen long-term rewards Λ_a and Λ_b , we can also estimate the expected long-term rewards Q_0 and Q_1 .

Although the eigen immediate/long-term rewards are just equivalent transformations of the expected immediate/long-term rewards, their iterative functions are much simplified. Actually, we have

PROPOSITION 3.3. [Iterative Function of Eigen Long-Term Rewards] The iterative function of the eigen long-term rewards writes

$$\begin{aligned}\Lambda_a(s_t, a_t) &= \Gamma_a(s_t, a_t) \\ \Lambda_b(s_t, a_t) &= \Gamma_b(s_t, a_t) + \gamma\Lambda_b(s_{t+1}, a_{t+1})\end{aligned}\quad (15)$$

The proof can be referred to in Appendix. A. We explain more about Proposition 3.3. Firstly, Λ_a is the difference between the long-term rewards under recommendations by real-time computation and by the cache, i.e., Q_0 and Q_1 . The first equation of Eq. (15) shows that such a difference equals the difference between the immediate rewards under real-time and cached recommendations. It is because the future cache states are stochastic and independent of the current cache state. Secondly, Λ_b , which is the weighted sum of real-time and cached recommendations, follows an iterative equation independent of Λ_a .

Compared to the iterative functions of the expected long-term rewards in Eq. (11) where the two critics depend on each other, the eigen long-term rewards Λ_a and Λ_b are independent. Therefore, if we regard the eigen long-term rewards as the function to learn, we will not suffer from the critic dependency problem.

Given the abovementioned discussions, we are ready to provide the final algorithm.

3.3 The Eigenfunction Learning Algorithm

Now, we provide the CARL-EL algorithm according to the eigenfunctions defined in Section 3.2. The framework of CARL-EL is shown in Figure 5. Specifically, we regard the following functions as learnable functions:

- The immediate reward functions $V_0(s_t, a_t; \psi_0)$ and $V_1(s_t, \psi_1)$, parameterized by ψ_0 and ψ_1 .
- The eigen long-term reward function $\Lambda_b(s_t, a_t; \rho)$, parameterized by ρ .
- The policy function $\mu(s_t; \theta)$, parameterized by θ .

In contrast, we regard the following functions as derived functions from the abovementioned learnable functions:

- The eigen immediate reward functions Γ_a and Γ_b , which can be obtained from the expected immediate reward V_0 and V_1 according to Eq. (12).
- The eigen long-term reward function Λ_a , which is equal to Γ_a according to Proposition 3.3.
- The expected long-term rewards Q_0 and Q_1 , which can be derived from the eigen long-term rewards Λ_a and Λ_b according to Eq. (14).

The EL algorithm, as shown in Algorithm 1, consists of five steps, i.e. learning the immediate rewards, calculating the eigen immediate rewards, learning the eigen long-term reward, recovering the long-term rewards, and learning the policy function.

Step 1: we learn the immediate reward functions $V_0(s_t, a_t; \psi_0)$ and $V_1(s_t, \psi_1)$. The loss function is

$$l(\psi) = \begin{cases} [r_t - V_0(s_t, a_t; \psi_0)]^2 & , C_t = 0 \\ [r_t - V_1(s_t, \psi_1)]^2 & , C_t = 1 \end{cases}\quad (16)$$

We learn different immediate rewards for requests processed by real-time and cached recommendations, where $C_t = 0$ or 1, respectively.

Step 2: after obtaining the estimation of immediate value functions, we calculate the eigen immediate rewards $\Gamma_a(s_t, a_t)$ and $\Gamma_b(s_t, a_t)$ according to Eq. (12).

Step 3: the eigen immediate rewards Γ_a and Γ_b are used to learn the eigen long-term rewards Λ_a and Λ_b via the temporal difference function. According to Proposition 3.3, we have $\Lambda_a = \Gamma_a$, therefore,

Algorithm 1 CARL-EL: Eigenfunction Learning of Cache-Aware Reinforcement Learning

```

1: Input:  $\{s_{1:T}, a_{1:T}, r_{1:T}\}$  for each user.
2: Output: A policy function  $\mu(s_t; \theta)$  parameterized by  $\theta$ .
3: for each user session with  $T$  requests from the replay buffer do
4:   for  $t = 1, \dots, T$  do
5:     Collect the user state  $s_t$ , the reward  $r_t$ , and the action  $a_t$  from the replay buffer.
6:     Step 1: Learn the immediate rewards  $V_0(s_t, a_t; \psi_0)$  and  $V_1(s_t; \psi_1)$  according to Eq. (16).
7:     Step 2: Calculate the eigen immediate rewards  $\Gamma_a(s_t, a_t)$  and  $\Gamma_b(s_t, a_t)$  according to Eq. (12).
8:     Step 3: Calculate the eigen long-term rewards  $\Lambda_a(s_t, a_t)$ , and learn  $\Lambda_b(s_t, a_t; \rho)$  according to Eq. (17).
9:     Step 4: Calculate the critic function  $Q_0(s_t, a_t)$  according to Eq. (14).
10:    Step 5: Take policy gradient with the policy loss  $Q_0(s_t, \mu(s_t; \theta))$  to update  $\theta$  according to Eq. (5).
11:   end for
12: end for

```

there is no need to learn Λ_a . In contrast, Λ_b needs to be learned by the following temporal difference loss:

$$l(\rho) = [\Lambda_b(s_t, a_t; \rho) - (\Gamma_b(s_t, a_t) + \gamma \Lambda_b(s_{t+1}, \mu(s_{t+1}; \theta^-); \rho^-))]^2 \quad (17)$$

where θ^- and ϕ^- are the parameters of the target actor and target critic, respectively.

Step 4: we recover the expected long-term reward Q_0 and Q_1 from the eigen long-term reward Λ_a and Λ_b according to Eq. (14).

Step 5: we regard the expected long-term reward $Q_0(s_t, a_t)$ as the policy loss, and take the policy gradient algorithm according to Eq. (5).

Figure 5 shows the information flow of Algorithm 1. Compared with the direct learning method shown in Figure 4, the main difference between Algorithm 1 and CARL-DL is the temporal difference equation, i.e. Eq. (17) in CARL-EL and Eq. (8) in CARL-DL. In CARL-DL, the learning of the critic functions Q_0 and Q_1 depend on each other according to a stochastic cache state C_t ; while in CARL-EL, the learning of the eigen long-term rewards Λ_a and Λ_b does not depend on each other. Therefore, the temporal difference of CARL-EL does not rely on the stochastic cache state C_t , and hence, the variance of learning can be effectively reduced, and the performance can be improved.

4 EXPERIMENTAL RESULTS

We deploy our proposed CARL model in Kwai, a short video platform serving over 100 million users. The QPS is shown in Figure 1, and the traffic router executes as described in Section 2. The ratio of cached recommendations during peak periods is about 40%. We did not conduct offline experiments because there is, to the best of the authors' knowledge, no offline experimental environment considering the impact of the cache for the time being.

4.0.1 Implementation. The structure of the online system is shown in Figure 6, and the details are as follows:

- **Sample Generation.** We first collect the states, actions, and rewards of each request to generate request-wise samples. Next, a session collector groups the requests of the same user and orders them by the timestamp of the requests. We split the request sample list into multiple sessions based on the 15-minute inactivity rule. The session collector constructs

two consecutive requests from the same session into a single RL sample and feeds them into the replay buffer.

- **MDP.** We deploy the actor in the ranking part of the recommender system. The recommender serves as the agent, and the users serve as the environment. At each user request, the actor gets the state s_t containing a vector of the user profile, the behavior history, the request context, and the statistics of candidate video features. The user profile covers the information collected in the registration, including the gender, the age, and the interests of the user. The behavior history includes the items that the user interacted with in the past. The request context includes the timestamp and the location, and the video statistics include the average score and the 10-th/30-th/50-th/70-th/90-th percentile of the prediction scores. Then, it returns an action vector a_t , which is a 5-dimensional continuous vector ranging in $[0, 3]$, acting as the fusion parameters of five scoring models predicting the main feedback (watchtime, shortview, longview, finish, and forward). A final ranking score is generated according to Eq. (2), of which the top L items are selected, and the first K items are shown to the users (the red arrow in Figure 6). The rest $L - K$ items are cached in case of a cached recommendation when the traffic exceeds the system's affordability (the green arrow in Figure 6). In our scenario, $L = 40$ and $K = 8$.

4.1 Settings

4.1.1 Baselines. We deploy different methods in the online system:

- **Cross Entropy Method (CEM)** [10]: A black-box optimization method commonly used for hyper-parameter optimization. We use CEM to search the best parameters a_t .
- **TD3** [3]: A famous RL method which uses twin critics to reduce the bias. TD3 does not explicitly distinguish the real-time and cached recommendations.
- **RLUR**[1]: A state-of-the-art reinforcement-learning-based method for the fusion of multiple predictions. RLUR is the last online version before CARL is deployed in Kwai. Similar to TD3, RLUR does not explicitly distinguish the real-time and cached recommendations, either.
- **CARL-DL:** The CARL with the direct learning method.
- **CARL-EL:** The CARL with the eigenfunction learning method.

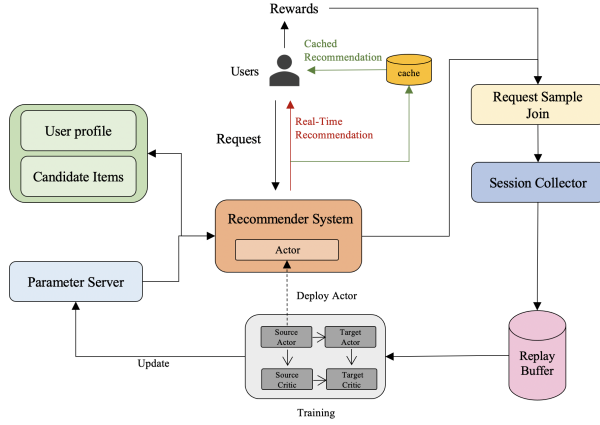


Figure 6: System Implementation

	Session Watch Time	Daily Watch Time
CEM	+0.000%	+0.000%
TD3	+0.176%	+0.184%
RLUR	+0.206%	+0.192%
CARL-DL	+0.390%	+0.342%
CARL-EL	+0.545%	+0.586%

Table 3: Experimental Results in Kwai in terms of the improvement over CEM.

4.1.2 Metrics. We evaluate the abovementioned algorithms by the session-wise watch time and the daily watch time, an important metric in short video platforms [14]. We show the experiment results in improvement compared to the CEM method.

4.2 Results

Table 3 shows the comparison results of different methods in online A/B experiments. In our scenario, even a 0.1% improvement in the watch time is significant. It is shown that RLUR and TD3 outperform CEM due to the effectiveness of RL approaches. Both the CARL-DL and the CARL-EL outperform RLUR and TD3, showing the effectiveness of the cache-aware modeling. Moreover, the CARL-EL achieves the best performance, showing the effectiveness of the proposed EL method.

4.3 Discussions

4.3.1 Improvement on Critic Loss. Table 4 shows the average critic loss of different RL-based methods over one day. We evaluate these methods' critic loss using the formula shown in Eq. (4). It is shown that CARL outperforms the other algorithms, showing the effectiveness of the explicit modeling of cache. Moreover, CARL-EL achieves a better critic loss than CARL-DL, showing that the EL approach effectively improves the performance of CARL.

4.3.2 Gaps between Recommendations by Real-Time Computation and by the Cache. Figure 7 shows the average long-term reward

Methods	TD3	RLUR	CARL-DL	CARL-EL
Loss	0.33	0.30	0.28	0.25

Table 4: Critic Loss of Different Methods.

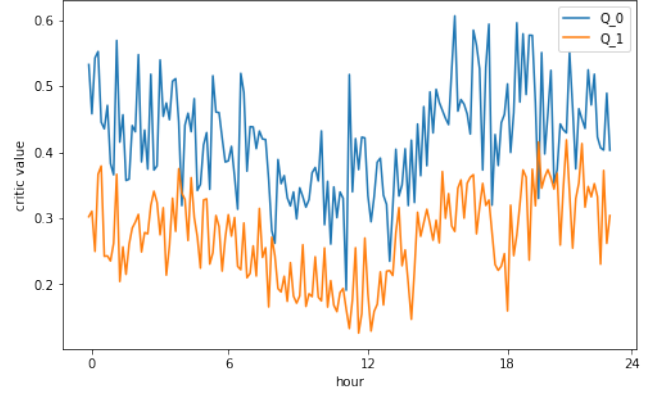


Figure 7: Comparisons of critic values.

estimation of the CARL-EL algorithm under recommendations by real-time computation and by the cache, i.e., Q_0 and Q_1 , respectively. It is shown that Q_0 is consistently larger than Q_1 , showing that the model successfully learns the advantage of real-time recommendations over cached recommendations. Moreover, Q_0 and Q_1 have similar trendings over one day. For example, they both reach minimal values at about 12pm, while reaching maximal values at about 18pm. It can be explained by Eq. (15), where $\Lambda_a = Q_0 - Q_1 = V_0 - V_1$, which means the difference between real-time and cached recommendations is independent of the cache ratios $D_0(t)$ and $D_1(t)$.

We further discuss the user feedback under recommendations by real-time computation and by the cache. Figure 8 shows the average watch time per video in recommendations by real-time recommendations and by the cache. Although the performance of cached recommendations is lower than that of real-time recommendations, the CARL model increases the performance of cached recommendations more than that of real-time recommendations. Such results show that effective cache modeling helps reduce the performance gap between real-time and cached recommendations.

4.3.3 Computational Burden. We test the total time cost of the recommender system under different methods. TD3 increases the time cost by 0.524% compared with CEM, while the difference in the time costs among these RL methods (TD3, RLUR, CARL-DL, and CARL-EL) is less than 0.1% because we keep actors unchanged in the experiments. Moreover, to verify the trade-off of computational burden and the recommendation performance introduced by the result cache, we test the RLUR in an experimental group without a result cache, i.e. all the requests are performed by real-time computation. Such settings increase the daily watch time by 0.642%, but with a significant increase of total time cost by 67.6%. In contrast, CARL-EL increases the daily watch time by 0.391% compared to

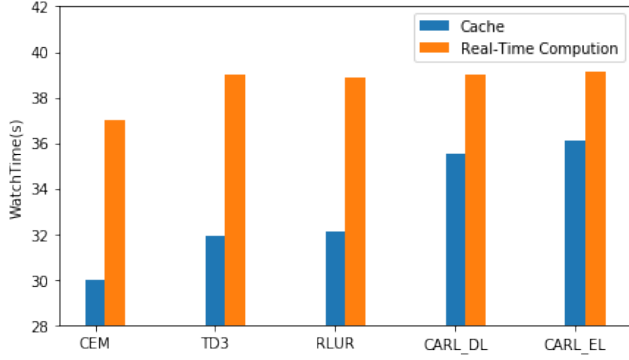


Figure 8: Average watch time per video of recommendations by real-time computation and by the cache.

RLUR with nearly no increase in the time cost, which shows the effectiveness of CARL-EL.

5 CONCLUSION

This paper proposes the CARL model to reinforce the long-term rewards in the recommender systems with a result cache. CARL uses a cache state to represent whether the recommender system performs recommendations by real-time computation or by the cache. The cache state is determined by the computational load of the recommender system. Moreover, we propose an EL algorithm to learn independent critics for CARL to solve the critic dependency. Experiments show that CARL can significantly improve the users' engagement when considering the result cache. CARL has been fully launched in Kwai app, serving over 100 million users.

REFERENCES

- [1] Qingpeng Cai, Shuchang Liu, Xueliang Wang, Tianyou Zuo, Wentao Xie, Bin Yang, Dong Zheng, Peng Jiang, and Kun Gai. 2023. Reinforcing User Retention in a Billion Scale Short Video Recommender System. *arXiv preprint arXiv:2302.01724* (2023).
- [2] Minmin Chen, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed H Chi. 2019. Top-k off-policy correction for a REINFORCE recommender system. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*. 456–464.
- [3] Scott Fujimoto, Herke Hoof, and David Meger. 2018. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*. PMLR, 1587–1596.
- [4] Chongming Gao, Shijun Li, Wenqiang Lei, Jiawei Chen, Biao Li, Peng Jiang, Xiangnan He, Jiaxin Mao, and Tat-Seng Chua. 2022. KuaiRec: A fully-observed dataset and insights for evaluating recommender systems. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 540–550.
- [5] Theodoros Giannakas, Pavlos Sermpezis, and Thrasyvoulos Spyropoulos. 2018. Show me the cache: Optimizing cache-friendly recommendations for sequential content access. In *2018 IEEE 19th International Symposium on "A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*. IEEE, 14–22.
- [6] Biye Jiang, Pengye Zhang, Rihan Chen, Xinchun Luo, Yin Yang, Guan Wang, Guorui Zhou, Xiaoqiang Zhu, and Kun Gai. 2020. DCAF: A Dynamic Computation Allocation Framework for Online Serving System. *arXiv preprint arXiv:2006.09684* (2020).
- [7] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data* 7, 3 (2019), 535–547.
- [8] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).
- [9] Shichen Liu, Fei Xiao, Wenwu Ou, and Luo Si. 2017. Cascade ranking for operational e-commerce search. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1557–1565.
- [10] Reuven Y Rubinfeld and Dirk P Kroese. 2004. *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation, and machine learning*. Vol. 133. Springer.
- [11] Yanfeng Wang, Mingyang Ding, Zhiyong Chen, and Ling Luo. 2017. Caching placement with recommendation systems for cache-enabled mobile social networks. *IEEE Communications Letters* 21, 10 (2017), 2266–2269.
- [12] Zhe Wang, Liqin Zhao, Biye Jiang, Guorui Zhou, Xiaoqiang Zhu, and Kun Gai. 2020. Cold: Towards the next generation of pre-ranking system. *arXiv preprint arXiv:2007.16122* (2020).
- [13] Wanqi Xue, Qingpeng Cai, Ruohan Zhan, Dong Zheng, Peng Jiang, and Bo An. 2022. ResAct: Reinforcing Long-term Engagement in Sequential Recommendation with Residual Actor. *arXiv preprint arXiv:2206.02620* (2022).
- [14] Ruohan Zhan, Changhua Pei, Qiang Su, Jianfeng Wen, Xueliang Wang, Guanyu Mu, Dong Zheng, Peng Jiang, and Kun Gai. 2022. Deconfounding Duration Bias in Watch-time Prediction for Video Recommendation. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 4472–4481.
- [15] Gengrui Zhang, Yao Wang, Xiaoshuang Chen, Hongyi Qian, Kaiqiao Zhan, and Ben Wang. 2024. UNEX-RL: Reinforcing Long-Term Rewards in Multi-Stage Recommender Systems with UNidirectional EXecution. *arXiv preprint arXiv:2401.06470* (2024).
- [16] Lixin Zou, Long Xia, Zhuoye Ding, Jiaxing Song, Weidong Liu, and Dawei Yin. 2019. Reinforcement learning to optimize long-term user engagement in recommender systems. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2810–2818.

A PROOF OF PROPOSITION 3.3

PROOF. We consider the iterative function Eq. (11) of the expected long-term rewards L_0 and L_1 . By taking the difference between the first equation and the second equation in Eq. (11), we have $L_0 - L_1 = V_0 - V_1$, i.e. $\Lambda_0 = \Gamma_0$. Then, by taking a weighted summation of the first and second equations in Eq. (11) with weight $D_0(t)$ and $D_1(t)$ respectively, we have

$$\begin{aligned}
 & D_0(t)L_0(s_t, \mathbf{a}_t) + D_1(t)L_1(s_t) \\
 &= D_0(t)V_0(s_t, \mathbf{a}_t) + D_1(t)V_1(s_t) \\
 &+ \gamma [D_0(t) + D_1(t)] D_0(t+1)\mathbb{E}_{\mathbf{a}_{t+1}} L_0(s_{t+1}, \mathbf{a}_{t+1}) \\
 &+ \gamma [D_0(t) + D_1(t)] D_1(t+1)\mathbb{E}_{\mathbf{a}_{t+1}} L_1(s_{t+1}, \mathbf{a}_{t+1})
 \end{aligned} \tag{18}$$

given the fact that $D_0(t) + D_1(t) = 1$, we have

$$\begin{aligned}
 & D_0(t)L_0(s_t, \mathbf{a}_t) + D_1(t)L_1(s_t) \\
 &= D_0(t)V_0(s_t, \mathbf{a}_t) + D_1(t)V_1(s_t) \\
 &+ \gamma \mathbb{E}_{\mathbf{a}_{t+1}} [D_0(t+1)L_0(s_{t+1}, \mathbf{a}_{t+1}) + D_1(t+1)L_1(s_{t+1}, \mathbf{a}_{t+1})]
 \end{aligned} \tag{19}$$

which completes the proof. $\square$