CARIS: Cache Affinity-aware Reinforced Intelligent Strategy

Yu Zuo[1], Yanlei Shang[1] ✉

Beijing University of Posts and Telecommunications, Beijing, China
anarionzuo@outlook.com,shangyl@bupt.edu.cn

Abstract. The cache is a critical component that directly impacts the computational system's performance and Quality of Service (QoS), with its effectiveness determined by the caching policy in use. An ideal caching policy must adapt to diverse workloads while approximating the optimal strategy. Recent advancements in caching strategies have focused on two paradigms: user access pattern prediction based on supervised learning and policy search methods based on Reinforcement Learning (RL). Supervised learning approaches often suffer from predictive inaccuracies. RL methods face challenges with large state spaces and insufficient exploration. Both paradigms lead to suboptimal performance and unbearable complexity in the decision making process. In this paper, we propose CARIS, a novel caching policy that integrates cache affinity estimation with a deep reinforcement learning agent. CARIS estimates the cache affinity of each user access, and focuses the agent's exploration on accesses with higher predicted cache affinity, effectively reducing unnecessary exploration and addressing the limitations of both paradigms. Additionally, to reduce complexity in decision making, CARIS introduces an "empty action" mechanism during cache hits to reduce noise from irrelevant actions. Through comprehensive experiments with multiple datasets, CARIS achieves an improvement more than 294% in the average cache hit rates compared to the SOTA learning-based caching policies under the tested scenarios. Our results highlight CARIS's potential to advance both theoretical understanding and practical efficiency in modern caching systems.

Keywords: Caching Strategy · Reinforcement Learning · Storage.

1 Introduction

The cache system is widely deployed in computer systems across various domains, greatly affecting their performance and quality of experience (QoE). The fundamental principle of the cache system is to store the frequently accessed data objects in a smaller but quicker-to-access medium, to reduce the access frequency on the larger but slower-to-access medium, so that the average access time is minimized. We find caches in networking systems, large databases, and micro systems on chips (SoCs).

Although caching greatly improves the system's performance in general, the key to its optimization is to find the most appropriate caching policy for the scenario under specific workloads. The classic heuristic-based caching policies are studied in depth, including Least-recently-used (LRU), Least-frequently-used (LFU), First-in-first-out (FIFO), and Greedy-dual-size-frequency (GDSF) [1]. They are simple to understand and implement, integrated in the commonly deployed cache systems (e.g. Redis [3], Memcached [7]). However, the storage system's access pattern is often complex and hard to predict, while these policies can only stick to their predetermined rules, not adapting to the evolving access pattern. The next step to developing better caching policies is to employ machine learning (ML) techniques.

Recently, many learning-based caching policies are proposed (e.g. [21], [10], [27], [22], [8]). Most of these works are based on the idea of predicting the user access pattern so as to approximate the globally optimal policy. For instance, the Belady's replacement policy always chooses the object that is accessed furthest to the future to replace when the cache is full and a cache load operation is imminent, making the key to its approximation the prediction of an object's revist time from the current moment. The recently proposed models predict patterns required by the theoretically optimal policies to invoke the ideal policies with confidence in their predictions. However, since all ML models suffer from inevitable limitations in their predicting abilities, these caching policies based on user access pattern prediction are always suboptimal.

Other policies based on reinforcement learning are also studied in depth (e.g. [15], [12]). Unlike the methods based on user access pattern predictions, the RL-based policies employ an agent to explore the cache system's state space and search for the optimal policy. By deciding on an objective such as maximizing object hit rate and using an RL training method (e.g. DQN [17], Actor-Critic [13], PPO [19]), the policy is learned in an end-to-end fashion, fulfilling the designated objective. The major limitation in the RL-based policies is that the state space is too large to be suffciently explored for an RL agent with a policy network that has a limited number of parameters in during the training phase. The agent is highly likely to spend an excessive amount of time on unimportant states and actions, resulting in its underperformance compared to the policies based on the user access pattern prediction.

For the problem of caching strategy optimization, the main challenges are (1) how the RL agent's exploration can be effectively simplified and (2) how cache affinity prediction can aid the agent's exploration. In this paper, we propose CARIS, a cache admission policy based on reinforcement learning and the cache affinity prediction on the user access, to address the aforementioned challenges. We summarize the key contributions of this work as follows:

1. CARIS introduces an "empty action" during cache hits. For a cache admission policy, the action taken in the event of a cache hit cannot actually influence the cache system's state, and the agent always receives a higher immediate reward. CARIS forces the agent to take the empty action on cache hit during training, not taking any irrelevant actions.

2. CARIS merges the concept of cache affinity into the RL policy. CARIS first estimates the cache affinity of each user access. Then the access deemed cache-averse is rejected to be admitted by the agent without consulting the major policy network. By filtering the cache-averse objects beforehand, the exploration focuses on objects that are more likely to generate cache hits in the near future.
3. We constructed datasets with real-world workloads to test CARIS and compare it with existing SOTA methods, and prove its superiority in terms of cache hit rate.

2 Related Works

This section introduces some basic concepts related to the following major sections, and recent research development concerning these concepts.

2.1 Deep Reinforcement Learning

A decision process is a sequence of states and actions of an agent, denoted as $\{(S_t, a_t)\}_{t=0}^{t=\infty}$. An agent is capable of making decisions at each step. If the state's probability distribution at time $t+1$ solely depends on the state and action taken at the previous moment t, we call the decision process Markovian, or a Markov Decision Process (MDP). That is, the state transition kernel has the form of $p(S_{t+1}|S_t, a_t)$.

At each moment in an MDP, an immediate reward depending on the state and action taken of the moment can be sampled by the agent, denoted as r_t and has the distribution of the form $p(r_t|S_t, a_t)$.

The agent must decide upon an action at each time t by some learned distribution $p(a_t|S_t)$ called policy, so the discounted sum of the rewards across all time steps is maximized. To learn such a policy, reinforcement learning methods are often employed. Common RL training methods based on deep learning (DL) includes DQN [4], Actor-Critic [13], TRPO, PPO [19], DDPG [25].

2.2 Caching Policies based on Access Pattern Prediction

Caching policies based on user access pattern prediction approximate the optimal policy by having the model predict a key variable derived from the user access sequences, that indirectly leads to optimality.

Multiple papers try to approximate the Belady's replacement policy. LRB [21] predicts the reuse time of each user-accessed object to meet Belady's requirement with GBM [11], and produce a cache replacement policy. LHR [27] predicts the hazard rate of the user-accessed objects, an upper bound of the reuse time, to do the same thing as in LRB [21]. Raven [8] does the estimation with a mixed-density neural network (MDN) [23].

Cache affinity of the user accesses has also been favored as a key to optimality. Hawkeye [10] trains a model to find the cache-averse objects, and replace them

with a higher priority. HR-Cache [22] employs the hazard rate estimation from LHR [27] as a measure of cache affinity, as in Hawkeye [10], and replaces them as in Hawkeye [10].

Methods based on user access pattern prediction have 2 major issues. (1) It requires explicit feature selection and engineering. (2) It suffers from predition inaccuracy.

CARIS resolves these issues by introducing an RL agent that is feature-agnostic and trained end-to-end. The agent explores in depth the cache system's state space induced by the accesses to the cache-friendly objects to mitigate the impact of the inaccurate predictions.

2.3 Caching Policies based on RL

The RL-based caching policies train an RL agent that outputs the action's probability distribution at each time step. Parrot [15] tries to approximate the Belady's cache replacement policy by imitation learning. RL-Cache [12] maximizes the discounted sum of the number of cache hits by Monte-Carlo sampling.

Different from Parrot [15] or RL-Cache [12], other studies do not present a caching policy for the general case, but for problems encountered in the specific fields. [14], [26], [2] constructed RL-based policy models for caching problems for networking in device-to-device scenarios, vehicles, video streaming and mobile edge respectively.

RL-based caching policies underperform policies based on access pattern prediction in general, due to the inefficiency of the agent's exploration. The more successful applications focus on special case scenarios, rather than the general cache system optimization problem.

CARIS resolves the exploration problem (1) by introducing an empty action during cache hits, and (2) by filtering away the cache-averse objects before the agent is invoked. By applying these two techniques, CARIS outperforms the existing RL-based methods without sharing any of their defectiveness.

3 The Caching Problem Analysis

This section first proves the caching policy optimization problem CARIS means to solve is equivalent to an RL problem in section 3.1. Then, sections 3.2,3.3 introduces the techniques used in solving the problem of interest.

3.1 The Cache Admission Process as an MDP

Consider a storage system with cache that recieves user access requests as time progresses. Denote the user-accessed data object at time t as e_t, and the cache table as c_t, that is the set of all objects in cache. We further consider a cache system with an admission policy, that decides whether the accessed object should be loaded into cache in the event of a cache miss. Denote the admission decision as a_t, with $a_t = 1$ meaning the object should be loaded into cache. Denote

the replaced data object in cache at time t as u_t. The cache behaviour can be described as equation 1, with the cache state unchanged when the admission policy rejects the object.

$$c_{t+1} = \begin{cases} c_t \cup \{e_t\} \backslash \{u_t\} & (a_t = 1) \\ c_t & (a_t = 0) \end{cases} \tag{1}$$

Let us first show that any caching decision process is Markovian. At any given time t, denote the sequence of the cache table c_t in equation 1 before time t as $C_t = \{c_\tau\}_{\tau=t}^{\tau=\infty}$, the cache admission action at time t as a_t, the user accessed object at time t as e_t, and the user's access sequence before time t as $U_t = \{e_\tau\}_{\tau=t}^{\tau=\infty}$. The cache table at time $t+1$ is determined by the cache table at time t according to equation 1. Consider the cache system's state as $S_t = (C_t, U_t)$. The next state S_{t+1} is determined once the current state S_t and action a_t is given, making the caching decision process Markovian.

Note that cache admission policy must be accompanied by a cache replacement policy, as in equation 1 u_t must be given when the cache is full. Any cache replacement policy can work in general.

The caching policy optimization problem can be formulated as an optimization problem of MDP. The cache system's state transition kernel is explicitly given in equation 2, where $p(C_{t+1}|C_t, e_{t+1}, U_t)$ is the δ function equivalent to equation 1.

$$p(S_{t+1}|S_t, a_t) = p(U_{t+1}, C_{t+1}|C_t, U_t, a_t) = p(e_{t+1}|U_t)p(C_{t+1}|C_t, e_{t+1}, U_t) \tag{2}$$

Denote $X_t = 1$ in the event of a cache hit, and $X_t = 0$ in the event of a cache miss. The optimization objective is to maximize the discounted number of cache hits, which is the discounted sum of X_t by a factor of γ less than 1, as in equation 3.

$$H_T = \sum_{t \geq T} \gamma^{t-T} X_t \tag{3}$$

Since the caching decision process can be considered Markovian, we propose to maximize the objective H_T in equation 3 by reinforcement learning methods for all choices of T. At any moment t, treat S_t as the agent's state, $a_t \in \{0, 1\}$ as the agent's action, and X_t sampled at time t as the immediate reward r_t. Denote the state space as $\mathbf{S}$, which is constructively formed by all possible values that S_t might take at any given moment t. H_T's expectation given the observed state at the same moment T is the state value function $V(S_T)$ by definition. By solving the reinforcement learning problem, CARIS finds the expected optimal H_T for all T of interest and the accompanied optimal caching policy.

Different from the general MDP problems, the decision process we consider yields immediate rewards regardless of the action taken at the moment, since whether the access attains a cache hit is not determined by whether the cache admits the accessed object. Therefore, during training, the agent can first sample states and rewards, then decides upon an action.

3.2 Focus on Cache Misses

The cache admission policy agent is not active when there is a cache hit. Only when there is a cache miss, is the policy's decision actually affecting the objective in equation 3. Based on this observation, CARIS introduces the empty action, denoted as $a_t = 2$. The empty action is equivalent to the no-admit action when there is a cache miss. CARIS is forced to take the empty action when there is a cache hit in training, as in equation 4, where $h_t = 0$ represents a cache miss at time t and $h_t = 1$ represents a cache hit.

$$p(a_t = 2|h_t = 0) = 0, p(a_t = 2|h_t = 1) = 1 \tag{4}$$

By the definition of the MDP optimization problem, if the action is either admit or no-admit, the immediate rewards and state transitions at that moment should be statistically dependent on the action, as in equation 2. By introducing the empty action, this assumption is satisfied for the admit and no-admit actions, and misleading exploration induced by actions during cache hits is avoided.

3.3 Focus on Cache-friendly Objects

Many objects are rarely revisted by the users within a short period of time, according to the dataset statistics in table 1a. As in Figure 2, the number of accesses for an object within one sequence is below 10 for the datasets used in the experiments. Generally speaking, the cache admission policy should adopt the behaviour that it admits with a higher probability the objects that are more likely to be repeatedly accessed by the users.

Denote $p^{m,R}_{T,T+L}$ for object m in some given time interval $[T, T + L)$ as the probability of the object being accessed R times in the given time interval, provided that the object is not replaced, as in equation 5. $\mathbf{M}^m_{T,T+L}$ is a collection of sequences that is composed of 0's and 1's and sums to R. $M \in \mathbf{M}^m_{T,T+L}$ is a sequence in the collection, and M_i denotes the ith object of the sequence. p^m_t is the probability of the accessed object e_t being object m. $p^{m,R}_{T,T+L}$ is small for most m, R, L.

$$p^{m,R}_{T,T+L} = \sum_{M \in \mathbf{M}^m_{T,T+L}} \prod_{t=T}^{T+L} (p^m_t)^{M_{t-T}} \tag{5}$$

To simplify the exploration, CARIS introduces the concept of cache affinity as in [10] and [27]. It considers an object m cache-friendly at time T if $p^{m,R}_{T,T+L} > \rho$ holds for some predetermined constants ρ, L, R, and the object is cache-averse if otherwise.

When there is a cache miss and CARIS is deciding whether it admits the accessed object, CARIS first evaluates its cache affinity. If the object is cache-averse, reject the object. If the object is cache-friendly, CARIS then consults the policy model trained as a reinforcement learning agent. By filtering out the cache-averse objects before the major reinforcement learning agent is taken into

account, the agent is protected from learning to decide on rarely revisted objects. The exploration is significantly simplified, and focuses only on the decisions concerning cache-friendly objects. The input space of the agent's policy distribution function $p(a_t|S_t)$ is reduced since the potential U_t in S_t is partially filtered away.

In section 4.2, we propose the cache affinity predictor as a part of CARIS to implement the aforementioned decision plan. In section 4.3, we adopt the maskable action from [9] to force the agent to perform a no-admit action in the event of the accessed object being cache-averse.

4 The CARIS Policy Model

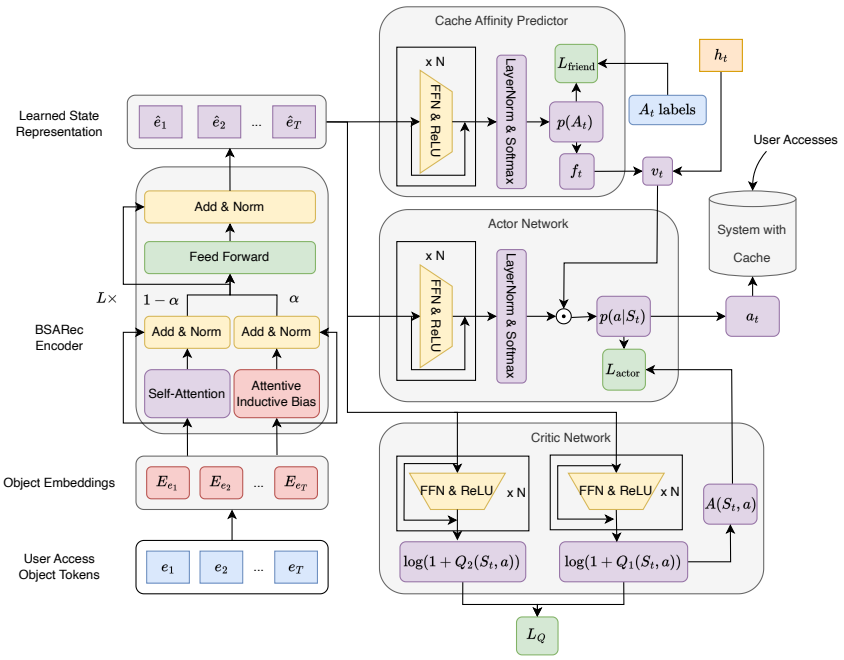


Fig. 1: Cache Affinity-aware Reinforced Intelligent Strategy Model

This section describes CARIS's internal structure in depth. CARIS is composed of a user access sequence encoder and multiple feed-forward neural networks, as in Figure 1, estimating the cache affinity, the state value function and the policy distribution. The hyper parameters used for model building and training are listed in table 2 in the appendix section A.

4.1 The Object Access Sequence Encoder

This subsection introduces the object access sequence encoder which learns the user-accessed objects' representation.

As in the cases of NLP and CV, the transformer familiy does a good job in learning the sequence representation for inputs from various domains. In our case, CARIS uses BSARec [20], a transformer-like architecture, to encode the object access sequence. The vanilla transformer suffers from information loss in the high-frequency spectrum of the representation computed at each of its layers. BSARec introduces a high-pass filter against this background to mitigate the problem.

As in Figure 1, the encoder based on BSARec takes the user access sequence tokens $e_1, ..., e_T$ as input, and produces vectors $\hat{e}_t$ as the representation $\hat{e}_t$ of the cache system state S_t.

4.2 The Cache Affinity Predictor

In this subsection, we propose to estimate the cache affinity from section 3.3. CARIS estimates $\hat{p}^m_{T,T+L}$ given by equation 6, an upper bound of $p^{m,R}_{T,T+L}$. $\hat{p}^m_{T,T+L}$ is the probability that object m is accessed at least 2 times between time T and $T+L$, so it satisfies 6. Since $p^{m,R}_{T,T+L} > 0$ for any m, R, T, L, we have $\hat{p}^m_{T,T+L} > p^{m,R}_{T,T+L}$, which is a tight bound when the probability that an object is revisted by a user is relatively small.

$$\hat{p}^m_{T,T+L} = 1 - \prod_{t=T}^{T+L}(1 - p^m_t) = \sum_{R \geq 2} p^{m,R}_{T,T+L} \tag{6}$$

For some predetermined constants $\rho \in (0, 1)$ and L, CARIS considers the object e_t accessed at time t cache-friendly if $\hat{p}^{e_t}_{T,T+L} \geq \rho$, and cache-averse if otherwise. At time t, denote the event that the obejct e_t is accessed in $(t, t+L)$ as $A_t = 1$, and $A_t = 0$ if otherwise. A_t's probability distribution is computed according to equation 7 based on the representation learned by the sequence encoder.

$$p(A_t = 1|\hat{e}_t) = \text{softmax}(\text{FFN}_{\text{affinity}}(\hat{e}_t)) \tag{7}$$

The loss function for the cache affinity predictor's training is equation 8, where we use i to differentiate the training experiences. Denote the number of state transitions in the training sequence l as $N^{(i)}_l$, and denote the number of experiences used in training as N_L.

$$L_{\text{affinity}} = -\frac{1}{N_L}\sum_i \frac{1}{N^{(i)}_l}\sum_t [A^{(i)}_t \log p(A^{(i)}_t = 1) + (1 - A^{(i)}_t)\log p(A^{(i)}_t = 0)] \tag{8}$$

4.3 The Actor Learner

This subsection describes the actor network and the computation of $p(a_t|S_t)$.

The policy distribution is computed based on the representation $\hat{e}_t$ given by the sequence encoder, as the actor network in Figure 1 and equation 9. The actor network outputs a 3 dimensional vector, representing the no-admit, admit, and empty actions respectively as $p(a_t|S_t)$.

$$p(a_t|S_t) = \text{softmax}(\text{FFN}_{\text{policy}}(\hat{e}_t)) \tag{9}$$

A valid action is the action that can be taken by the agent. CARIS computes the action validity indicators, with which the policy rules described in sctions 3.2,3.3 are enforced.When there is a cache miss and $\hat{p}^m_{T,T+L} \geq \rho$, $a_t = 0$ is an invalid action, while other actions are valid. When there is a cache hit, the only valid action is $a_t = 2$. Denote the event that the action i at time t is valid as $v^i_t = 1$, and as $v^0_t = 0$ if otherwise. Denote the event that there is a cache hit at time t as $h_t = 1$, and $h_t = 0$ if otherwise. Denote the event that $\hat{p}^{e_t}_{T,T+L} < \rho$ as $f_t = 1$, and $f_t = 0$ if otherwise. The validity indicators v^i_t for each of the actions i can be computed as in equation 10.

$$\begin{cases} v^0_t &= (1-h_t)f_i \\ v^1_t &= 1-h_t \\ v^2_t &= h_t \end{cases} \tag{10}$$

CARIS adopts action masks [9] to implement the action constraints. To avoid taking an invalid action, the input logits to the softmax function in equation 9 is set to a very small negative constant (CARIS uses -10^8) to represent $-\infty$, so that the component produce 0 probability. Reformulate equation 9 to 11, where $\odot$ is the vector's element-wise product.

One may consider enforcing equation 4 by assigning negative rewards to invalid actions during training. However, the agent would still have to spend time exploring by taking invalid actions and might not come to the right conclusion after lots of efforts. This is explained in [9] in detail. By using the action masks, we help the agent skip the exploring part.

$$p(a_t|S_t) = \text{softmax}(v_t \odot \text{FFN}_{\text{policy}}(\hat{e}_t) + (1-v_t)\cdot(-\infty)) \tag{11}$$

CARIS uses the PPO algorithm to train the policy network. Once a state transition from S_t to S_{t+1} by action a_t is observed, the loss is given by equation 12, where $R_t = \frac{p(a_t|S^{(i)}_t)}{\mathbf{sg}[p(a_t|S^{(i)}_t)]}$, and $\mathbf{sg}$ is the stop-gradient operator in the computation graph. $A(S_t, a_t)$ is the estimate of the advantage, which is discussed in section 4.4 in depth. CARIS introduces a penalty on the policy's entropy $\mathbf{H}[p(\cdot|S^{(i)}_t)]$, preventing the policy from collapsing onto a single action too early.

$$(L_{actor})^{(i)}_t = -\min\left[R_t A(S^{(i)}_t, a_t), \text{clip}\left(R_t, 1+\epsilon, 1-\epsilon\right)\right] - \beta_1 \mathbf{H}[p(\cdot|S^{(i)}_t)] \tag{12}$$

4.4 The Critic Learner

This subsection describes the estimation of $A(S_t, a_t)$ required by equation 12. For any possible state S_t and action a, CARIS estimates the state value function $V(S_t)$ and state-action value function $Q(S_t, a)$, then derives the advantage estimate by computing $A(S_t, a) = Q(S_t, a) - V(S_t)$. The state value function can be estimated by Q's estimate, as $V(S_t) = \max_a Q(S_t, a)$.

The Q function is estimated by the critic network part in Figure 1. Like DreamerV3 [5], the multilayered feed-forward network's output is set to be the symlog value of Q's estimate, as in equation 14, where the symlog function and its inverse symexp is given in equation 13.

$$\text{symlog}(x) = \log(1 + x), \text{symexp}(x) = \exp(x) - 1 \tag{13}$$

$$\text{symlog}(Q(S_t, a)) = \text{FFN}_{\text{Q}}(\hat{e}_t) \tag{14}$$

The critic network's parameters are trained in the double DQN fashion. The critic network is composed of 2 multilayered feed-forward networks, denoted as the training and target critics respectively, as in Figure 1. Denote the output of the training critic as $Q_1(S_t, a)$ and the output of the target critic as $Q_2(S_t, a)$. Once a state transition from S_t to S_{t+1} by action a_t is observed, the loss given by equation 16 is computed and accumulated into the total loss. In equation 15, r_t is the immediate reward yielded by the state transition, and **sg** is the stop-gradient operator in the computation graph, so no parameter update would be performed on its input.

$$\hat{Q}(S_t, a_t) = r_t + \gamma \mathbf{sg}[Q_2(S_{t+1}, \arg\max_a Q_1(S_{t+1}, a))] \tag{15}$$

$$L_Q = \frac{1}{N_L} \sum_{i,t} \frac{1}{2} \left[\text{symlog}(Q_1(S_t^{(i)}, a_t^{(i)})) - \text{symlog}(\hat{Q}(S_t^{(i)}, a_t^{(i)})) \right]^2 \tag{16}$$

Combining equations 8,16,12, the total loss is the weighted sum of the 3 losses, as in equation 17, where β_2, β_3 are the predetermined weights.

$$L = L_{actor} + \beta_2 L_Q + \beta_3 L_{\text{affinity}} \tag{17}$$

4.5 Offline Training and Online Inference

CARIS is trained in the same way as the vanilla PPO algorithm, in an on-policy manner with a replay buffer as in [4]. Training experiences are generated by a cache system simulation and fed into the replay buffer. The training process samples experiences from the replay buffer and performs parameter updates.

When using a trained CARIS model for online inference, CARIS splits the user access sequence into 2 subsequencesses by picking a seperating time step. The moments before the seperating time step t is denoted as the history window,

while the moments after the seperating time step t is denoted as the inference window. After an inference operation is performed, the cache system performs cache loading operations according to the inference results of the items in the inference window. Then, user access records in the inference window are merged into the history window, and the inference window is cleared. New user access records are added to the empty inference window, before the next inference operation clears it again. When the history window grows larger than a predetermined limit, the window removes the oldest user access records each time new records are added. By adjusting the window sizes, the trade-off between inference time and system performance is addressed.

To simulate the data distribution in inference, the training dataset is expanded. For each sequence $\{e_t\}_{t=0}^{t=N^{(i)}}$ used during training, the expanded sequences are created according to $\{\{e_t\}_{t=0}^{t=l} | \forall l \in [1, N^{(i)}]\}$, to form a step-by-step scenario. All training sequences are expanded in this fashion to construct an expanded sequence set. The sequences used to generate experiences are sampled from the exapneded sequence set.

Our model deployed in an real world application system is updated every hour. Newly received user access records are appended to the ever-growing training dataset, which is then used to incrementally train the model.

5 Experiments

Our experiments on CARIS were conducted with the PyTorch library [18]. A cache system simulator was built to evaluate various kinds of caching policies. Our code and dataset is publicly available at GitHub[1].

5.1 Datasets and Cache System Settings

We used the datasets enumerated in table 1a for simulation and policy evaluation. The datasets have diverse characteristics. (1) The ML-1M [6] dataset contains user rating records on movies. We sort the rating records with respect to their timestamp for each user. Consider the movies as data objects and the rating records as a user's access to that object. The dataset is reformulated as a collection of user access sequences. (2) The Wiki dataset [24] is the access log of the Wikipedia website's server. We extract from it the requests to the main pages in Wikipedia, and divide the log into subsequences of length no larger than 550 to form a dataset of user access sequences. (3) The Twitter dataset [28] is the access log of Twitter's cache server. We sort the access records with respect to the access time for each user, like that is done with the ML-1M dataset. The dataset is reformed into a collection of user access sequences. We also tested CARIS with real-world workloads. While the results supported our work, these details cannot be disclosed due to commercial privacy and proprietary restrictions.

Table 1a includes some statistics of these datasets. The meaning of each of the statistics is listed here. (1) #Items is the total number of data objects accessed

[1] `https://github.com/Anarion-zuo/caris_repo`

Table 1: Datasets

(a) Datasets Statistics

Dataset		ML-1M	Wiki	Twitter
#Items		3416	12236	8280
Frequency	Avg	292.63	101.24	1207.73
	P50	146	104	50
	P90	765	138	2406
#Distinct	Avg	624.74	467.2	770.38
	P50	563	467	772
	P90	874.6	478	927
LEN	Avg	751.59	549.08	1665.83
	P50	677	549	1618
	P90	1040.6	549	2290.8
OHWR		0.69	0.81	0.25

(b) Cache Item Capacity

Dataset	Cache Capacity
ML-1M	17, 34, 68, 102
Wiki	13, 25, 50, 75
Twitter	75, 100, 125, 150

by the user. (2) Frequency is the number of accesses to each data object. (3) Distinct is the number of distinct data objects in each user access sequence. (4) LEN is the length of each user access sequence. (5) OHWR is the ratio between the number of data objects that are accessed only once in each user access sequence, and the number of user accesses in that same sequence.

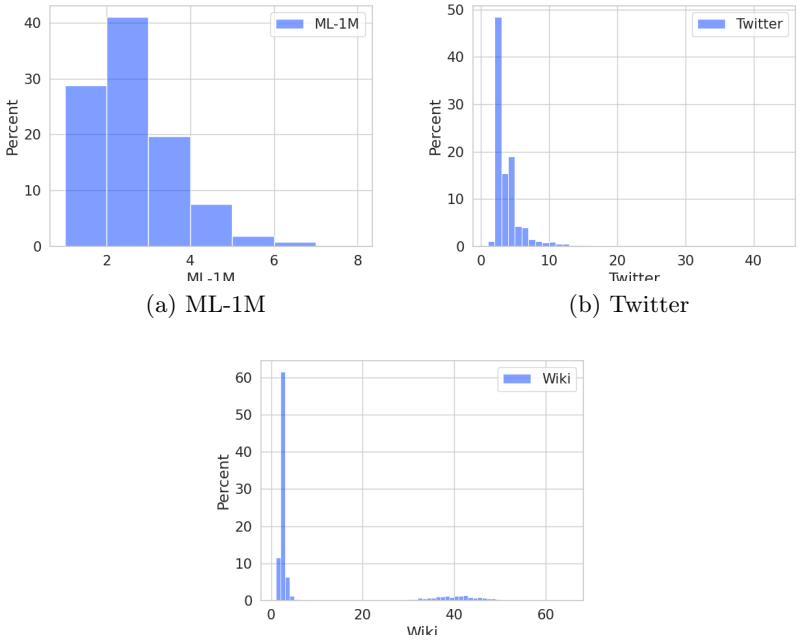
Figure 2 depicts the distribution of the number of accesses for each data object within one sequence. Most objects are rarely accessed more than 10 times within one sequence.

A good caching strategy must perform well under difficult conditions, so the simulated cache capacities were chosen to be no more than 3% of the number of all data objects in each dataset, as given in table 1b.

5.2 Experimental Methodology

Compared Caching Policies We compared CARIS with 10 different caching policies, including Random, FIFO, LRU, LRU-4, LFU, GDSF [1], PredMarker [16], LRB [21], Raven [8], HR-Cache [22]. Random is the random replacement policy, which randomly selects an object in the cache to replace when the cache is full. PredMarker [16], LRB [21], Raven [8], HR-Cache [22] are policies based on machine learning, while others are based on heuristic rules.

Evaluation Metrics For each dataset, randomly pick 64 sequences for performance evaluation, and use the rest for training. For each of the sequences used in evaluation, the last 100 user accesses were used as the inference window, as in section 4.5. We used the cache object hit rate (OHR) in the inference window as the performance metric of a evaluation sequence. To enable comparison across datasets, we used the OHR of the Random policy as a baseline value



(a) ML-1M (b) Twitter

(c) Wiki

Fig. 2: Histograms of the Objects' #Accesses within One Sequence

and computed the normalized metric score with $x_r = \frac{x}{x_{\text{base}}} - 1$. x_{base} was set to be the OHR of the Random policy given the testing conditions if not specified otherwise.

5.3 Main Results

Figure 3 depicts the CARIS's OHR (not normalized) on the 3 datasets we used under various simulated cache capacities from table 1b. Figure 4 depicts the normalized OHR of other policies compared to CARIS's, with the Random policy's OHR as the baseline value x_{base}.

Under all experimented conditions, CARIS outperformed all other policies consistently. Compared to the learning-based methods, CARIS achieved an improvement of more than 294% on average. CARIS's advantage differed on the various datasets, with ML-1M being the most victorious. Note that CARIS's normalized OHR did not monotonically increase with the simulated cache capacity, while its OHR did.

The compared learning-based methods (PredMarker, LRB, Raven, HR-Cache) exhibited apparent drawbacks. They are policies based on user access pattern predictions, whose performance is largely affected by the dataset's one hit wonder ratio. When the one hit wonder ratio was large, wrong predictions were less

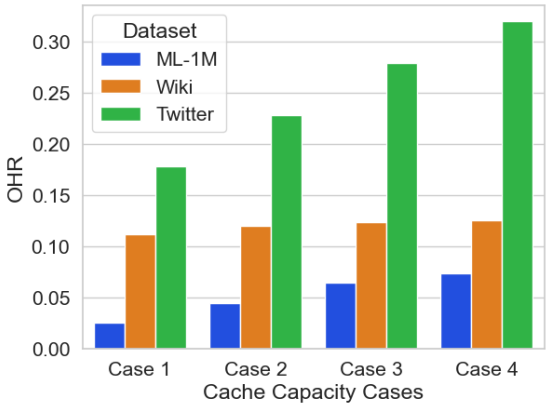
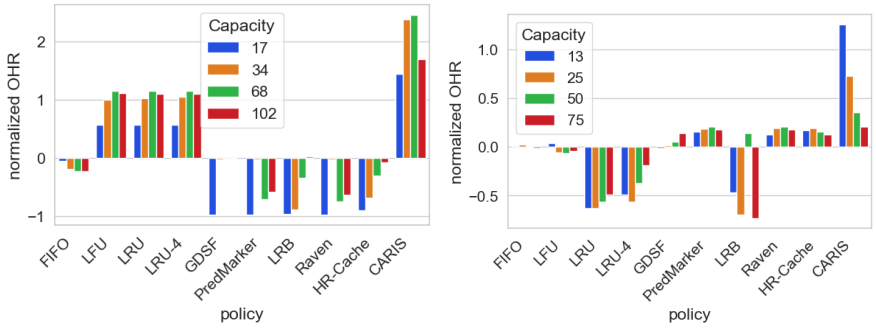
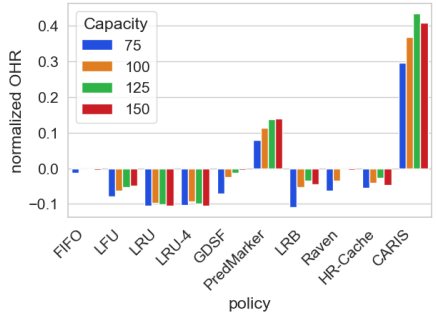


Fig. 3: CARIS OHR under Various Dataset and Cache Capacities



(a) ML-1M

(b) Wiki



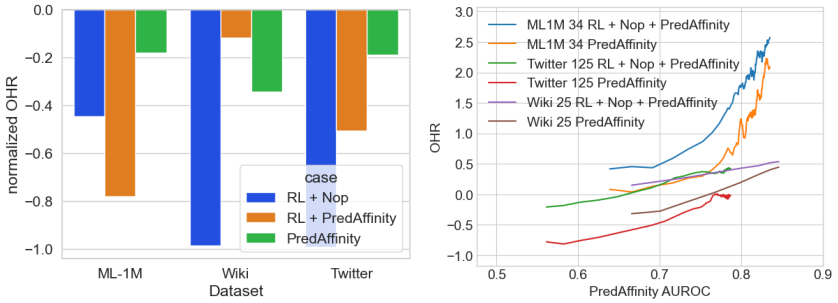
(c) Twitter

Fig. 4: CARIS OHR Compared with Other Policies

harmful, since a right prediction hardly pays off when objects are rarely revisted. The Wiki dataset has the largest one hit wonder ratio, and these policies performed the best and were closest to CARIS on Wiki.

However, when the cache capacity was relatively small, the prediction-based methods suffered more from a bad decision, since cache hits were relatively rare. Therefore they were even outperformed by the Random policy. CARIS has an RL agent to mitigate the damage caused by prediction errors.

5.4 Ablation Study



(a) OHR when Key Features are Removed

(b) PredAffinity AUROC vs OHR during Training

Fig. 5: Ablation Study Results

Consider the ablation study results in Figure 5. Denote the CARIS without the empty action as RL+PredAffinity, and the case without the affinity predictor as RL+Nop. Denote the CARIS without the RL agent as PredAffinity, using soley the affinity predictor for object admission decisions. The PredAffinity CARIS admits objects when the access is considered cache-friendly. Figure 5 depicts the normalized OHR of CARIS when these key features were removed from it. The simulated cache capacity for the 3 datasets were 34, 100, 100 for these ablation studies. The baseline value x_{base} used in score rescaling was the OHR of the original CARIS. As is demonstrated by Figure 5, any removal of the key components in CARIS damaged it.

A simple observation showed the importance of the cache affinity predictor in CARIS. Degradation was worst in the RL+Nop case on datasets Wiki and Twitter, where their OHR was so small compared to the original CARIS that the normalized scores were close to -1. However, since Twitter had its one hit wonder ratio smaller than Wiki's, and RL+Nop degraded almost equally on both datasets, the cache affinity is equally imoprtant for user access patterns with both larger or smaller probability of revisiting the previously accessed objects.

We further investigated the impact of the cache affinity predictor's accuracy on CARIS's performance. We tested on the datasets ML-1M, Wiki and Twitter with simulated cache capacities 34, 25, 125, and depict the relation between the prediction's AUROC score and the achieved OHR of CARIS during the training process in Figure 5. As before, the PredAffinity case was outperformed by the

fully equipped CARIS throughout the training process. Obviously, OHR got higher as the AUROC got larger and the prediction became more accurate.

6 Conclusions

In this paper, we introduce the caching admission policy model CARIS, based on RL for the most part, and aided by cache affinity estimation. CARIS uses a Transformer-like encoder [20] to learn the sequence item representation, based on which it computes the predicted cache affinity and the policy. By focusing the exploration on the cache-friendly user accesses, CARIS's training process is significantly simplified. Besides, CARIS is forced to take an empty action in the event of a cache hit, addressing the problem of useless caching actions. To the best of our knowledge, CARIS is the first caching policy model that incorporates RL with prediction methods based on supervised learning.

Presently, the CARIS model is deployed in our web application servers. We are looking to generalize its application to other systems such as CDN and edge computing, so as to verify its ability across diverse fields.

References

1. Arlitt, M., Cherkasova, L., Dilley, J., Friedrich, R., Jin, T.: Evaluating content management techniques for web proxy caches. SIGMETRICS Perform. Eval. Rev. **27**(4), 3–11 (Mar 2000)
2. Cui, L., Ni, E., Zhou, Y., Wang, Z., Zhang, L., Liu, J., Xu, Y.: Towards real-time video caching at edge servers: A cost-aware deep q-learning solution. IEEE Transactions on Multimedia **25**, 302–314 (2023)
3. Eddelbuettel, D.: A brief introduction to redis (2022)
4. Hafiz, A.M.: A survey of deep q-networks used for reinforcement learning: State of the art. In: Intelligent Communication Technologies and Virtual Mobile Networks. pp. 393–402 (2023)
5. Hafner, D., Pasukonis, J., Ba, J., Lillicrap, T.: Mastering diverse domains through world models (2024)
6. Harper, F.M., Konstan, J.A.: The movielens datasets: History and context. ACM Trans. Interact. Intell. Syst. **5**(4) (Dec 2015)
7. Hei, W., Sun, B., Wang, Q.: A memcached cache-based approach to fast response for large language models in power systems. In: Proceedings of the 2024 International Conference on Computer and Multimedia Technology. p. 547–553. ICCMT '24 (2024)
8. Hu, X., Ramadan, E., Ye, W., Tian, F., Zhang, Z.L.: Raven: belady-guided, predictive (deep) learning for in-memory and content caching. In: Proceedings of the 18th International Conference on Emerging Networking EXperiments and Technologies. p. 72–90. CoNEXT '22 (2022)
9. Huang, S., Ontañón, S.: A closer look at invalid action masking in policy gradient algorithms. The International FLAIRS Conference Proceedings **35** (May 2022)
10. Jain, A., Lin, C.: Back to the future: Leveraging belady's algorithm for improved cache replacement. In: 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA). pp. 78–89 (2016)

11. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.Y.: Lightgbm: A highly efficient gradient boosting decision tree. In: Advances in Neural Information Processing Systems. vol. 30 (2017)
12. Kirilin, V., Sundarrajan, A., Gorinsky, S., Sitaraman, R.K.: Rl-cache: Learning-based cache admission for content delivery. In: Proceedings of the 2019 Workshop on Network Meets AI & ML. p. 57–63. NetAI'19 (2019)
13. Kumar, H., Koppel, A., Ribeiro, A.: On the sample complexity of actor-critic method for reinforcement learning with function approximation. Machine Learning **112**(7), 2433–2467 (2023)
14. Li, L., Xu, Y., Yin, J., Liang, W., Li, X., Chen, W., Han, Z.: Deep reinforcement learning approaches for content caching in cache-enabled d2d networks. IEEE Internet of Things Journal **7**(1), 544–557 (2020)
15. Liu, E., Hashemi, M., Swersky, K., Ranganathan, P., Ahn, J.: An imitation learning approach for cache replacement. In: Proceedings of the 37th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 119, pp. 6237–6247 (13–18 Jul 2020)
16. Lykouris, T., Vassilvtiskii, S.: Competitive caching with machine learned advice. In: Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 3296–3305 (10–15 Jul 2018)
17. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
18. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: PyTorch: an imperative style, high-performance deep learning library. Curran Associates Inc. (2019)
19. Rio, A.d., Jimenez, D., Serrano, J.: Comparative analysis of a3c and ppo algorithms in reinforcement learning: A survey on general environments. IEEE Access **12**, 146795–146806 (2024)
20. Shin, Y., Choi, J., Wi, H., Park, N.: An attentive inductive bias for sequential recommendation beyond the self-attention (2024)
21. Song, Z., Berger, D.S., Li, K., Lloyd, W.: Learning relaxed belady for content distribution network caching. In: 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20). pp. 529–544 (Feb 2020)
22. Torabi, H., Khazaei, H., Litoiu, M.: A learning-based caching mechanism for edge content delivery. In: Proceedings of the 15th ACM/SPEC International Conference on Performance Engineering. p. 236–246. ICPE '24, ACM (May 2024)
23. Tosi, F., Liao, Y., Schmitt, C., Geiger, A.: Smd-nets: Stereo mixture density networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 8942–8952 (June 2021)
24. Urdaneta, G., Pierre, G., van Steen, M.: Wikipedia workload analysis (10 2007)
25. Wang, X., Wang, S., Liang, X., Zhao, D., Huang, J., Xu, X., Dai, B., Miao, Q.: Deep reinforcement learning: A survey. IEEE Transactions on Neural Networks and Learning Systems **35**(4), 5064–5078 (2024)
26. Wu, Q., Zhao, Y., Fan, Q., Fan, P., Wang, J., Zhang, C.: Mobility-aware cooperative caching in vehicular edge computing based on asynchronous federated and deep reinforcement learning. IEEE Journal of Selected Topics in Signal Processing **17**(1), 66–81 (2023)

27. Yan, G., Li, J., Towsley, D.: Learning from optimal caching for content delivery. In: Proceedings of the 17th International Conference on Emerging Networking EXperiments and Technologies. p. 344–358. CoNEXT '21 (2021)
28. Yang, J., Yue, Y., Rashmi, K.V.: A large scale analysis of hundreds of in-memory cache clusters at twitter. In: 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20). pp. 191–208 (Nov 2020)

A Parameters Used for Model and Training

Table 2: Model & Training Parameters

Parameter Name	Chosen Value
Learning rate	0.0005
Random seed	42
BSARec α	0.7
BSARec c	5
Batch size	16
BSARec attention #heads	4
BSARec hidden #layers	6
Representation hidden size	64
Adam beta 1	0.9
Adam beta 2	0.999
Adam weight decay	0.1
Actor network #layers	2
Critic network #layers	2
BSARec hidden layers dropout	0.5
BSARec attention dropout	0.5
Actor network dropout	0.5
Critic network dropout	0.5
Discount ratio γ	0.99
Replay buffer size	1024
Entropy loss weight β_1	1
Q loss weight β_2	1
Cache affinity loss weight β_3	1
Affinity threshold ρ	0.3
Cache affinity range L	400
PPO clip tolerance ϵ	0.1

Table 2 lists the parameters used in deep learning model building and training. The parameters prefixed with "BSARec" are for the encoder part of the model based on BSARec [20].