Computer Modeling in Engineering & Sciences

Tech Science Press

DOI: 10.32604/cmes.2024.048723

ARTICLE

Proactive Caching at the Wireless Edge: A Novel Predictive User Popularity-Aware Approach

Yunye Wan[1], Peng Chen[2], Yunni Xia[1,*], Yong Ma[3], Dongge Zhu[4], Xu Wang[5], Hui Liu[6], Weiling Li[7], Xianhua Niu[2], Lei Xu[8] and Yumin Dong[9]

[1]College of Computer Science, Chongqing University, Chongqing, 400044, China

[2]School of Computer and Software Engineering, Xihua University, Chengdu, 610039, China

[3]School of Computer and Information Engineering, Jiangxi Normal University, Nanchang, 330022, China

[4]Electric Power Research Institute of State Grid Ningxia Electric Power Co., Ltd., Yinchuan, 750002, China

[5]College of Mechanical and Vehicle Engineering, Chongqing University, Chongqing, 400030, China

[6]School of Computer Science and Technology, Beijing Institute of Technology, Beijing, 100083, China

[7]School of Computer Science and Technology, Dongguan University of Technology, Dongguan, 523808, China

[8]School of Emergency Management, Xihua University, Chengdu, 610039, China

[9]College of Computer and Information Science, Chongqing Normal University, Chongqing, 401331, China

*Corresponding Author: Yunni Xia. Email: xiayunni@hotmail.com

Received: 16 December 2023 Accepted: 07 March 2024 Published: 20 May 2024

ABSTRACT

Mobile Edge Computing (MEC) is a promising technology that provides on-demand computing and efficient storage services as close to end users as possible. In an MEC environment, servers are deployed closer to mobile terminals to exploit storage infrastructure, improve content delivery efficiency, and enhance user experience. However, due to the limited capacity of edge servers, it remains a significant challenge to meet the changing, time-varying, and customized needs for highly diversified content of users. Recently, techniques for caching content at the edge are becoming popular for addressing the above challenges. It is capable of filling the communication gap between the users and content providers while relieving pressure on remote cloud servers. However, existing static caching strategies are still inefficient in handling the dynamics of the time-varying popularity of content and meeting users' demands for highly diversified entity data. To address this challenge, we introduce a novel method for content caching over MEC, i.e., PRIME. It synthesizes a content popularity prediction model, which takes users' stay time and their request traces as inputs, and a deep reinforcement learning model for yielding dynamic caching schedules. Experimental results demonstrate that PRIME, when tested upon the MovieLens 1M dataset for user request patterns and the Shanghai Telecom dataset for user mobility, outperforms its peers in terms of cache hit rates, transmission latency, and system cost.

KEYWORDS

Mobile edge computing; content caching; system average cost; deep reinforcement learning; collaborative mechanism

1 Introduction

The rapid growth of the Internet of Things (IoT) [1] has spurred the development of MEC [2], a field that has garnered significant attention in the domains of information technology and communication. Traditional mobile networks are ineffective in meeting crucial requirements, such as minimizing latency and content transmission costs due to the distance between service providers and users. The MEC paradigm addresses these challenges by providing robust computing capabilities at Internet access points close to users and offloading computing tasks from remote cloud servers to physical network edge nodes. This approach significantly reduces data transmission latency and achieves high energy efficiency, enabling real-time responsive applications and sensitivity-requiring services.

Edge content caching is an important component of MEC, which further improves the efficiency of resource storage and delivery. The core idea of this technology is to cache frequently accessed content on edge servers close to users to guarantee high responsiveness when content requests arrive and alleviate the traffic from/to remote cloud data centers. This approach improves systems responsiveness and guarantees low latency in content delivery. Consequently, it achieves high quality of experience (QoE) for content requestors as well as a significant portion of requested content can be delivered from nearby storage.

However, existing solutions to edge content caching still need to be improved in several ways. On one hand, the storage capacity of edge servers is usually limited. Given the high user mobility in mobile edge networks [3], previously in-demand content can quickly become outdated. Consequently, static caching schemes often fail to meet content requests when users in MEC are with high mobility. On the other hand, the users' preference for content demand can also be time-varying in spatial and temporal domains. Content providers are thus supposed to smartly change the caching schemes over time to accommodate such changes to guarantee high hit rates of content.

To address the above challenges, we propose a novel predictive popularity-aware approach for proactive MEC caching, PRIME. PRIME aims to predict the popularity of content among users and cache content over MEC servers in a dynamic MEC environment. PRIME is featured by:

(i) It synthesizes a content popularity prediction model with a deep reinforcement learning model for capturing user mobility and content popularity.

(ii) It is capable of forecasting future content popularity and yield dynamic caching schedules accordingly.

The structure of this paper is organized into the following sections: Section 2 provides a review of related research, Section 3 introduces the system model and problem definition, Section 4 describes the proposed method, and Section 5 presents the performance evaluation results.

2 Related Work

With the booming of 5G and MEC technologies, content caching has gained significant attention in academic research in recent years. Particularly in 5G infrastructures using mmWave Massive MIMO systems [4], the constraints on storage, communication, and computational capacities at base stations pose significant challenges due to the increase in latency-sensitive applications. Content caching reduces data access latency, enhances service quality, and decreases core network load by storing popular content closer to users at network edges, thereby improving overall network energy efficiency. The most widely used methods are Least Recently Used (LRU) [5] and Least Frequently

Used (LFU) [6], which optimize content storage and management by intelligently deciding caching schedules in terms of plans of content deployment and replacement, at run time.

To further improve storage efficiency in MEC, recently, proactive caching strategies have gained attention [7]. These strategies aim to proactively cache popular content near users by analyzing user preferences. For instance, Garg et al. [8] proposed a strategy for handling unknown and changing content popularity. It utilizes an online prediction method for analyzing the difference in average successful probability (ASP) and an online learning method for minimizing mean squared error and ASP regret. Li et al. [9] considered user mobility by distinguishing between fast and slow-moving users and used a Long Short-Term Memory (LSTM) network for predicting content popularity. Yu et al. [10] leveraged a federated learning model for identifying user interests. Similarly, Qi et al. [11] leveraged a federated learning framework for handling weighted aggregation of personal preferences. Zhang et al. [12] introduced the PSAC algorithm, which employs a self-attention mechanism for reducing network load. It pre-caches content at network edges based on user preferences and further analyzes sequential characteristics of user requests for predicting and re-caching content at the edge. Gao et al. [13] developed a token bucket-based dynamic batching (TBDB) algorithm that dynamically adjusts the maximum batch size (MBS) of cached content for optimizing device utilization and reducing system load. Additionally, Uthansakul et al. [14] designed a hybrid analog/digital precoder and decoder, proposing an alternating optimization algorithm to improve system energy efficiency and compute optimal system parameters. Wei et al. [15] proposed the SAPoC algorithm efficiently manages energy use by determining content popularity based on historical requests and similarities to popular existing content. Gao et al. [16] introduced a neural collaborative sequential learning mechanism deriving sequential information from biased user behavior sequences. Tang et al. [17] developed the Caser method. It transforms user interactions into low-dimensional embeddings and utilizes Convolutional Neural Networks (CNNs) for analyzing local interaction patterns of content requests.

Recently, learning-based methods and algorithms are becoming popular in related works [18]. These models learn to yield intelligent caching schedules through interaction with the environment. For instance, Cai et al. [19] considered user mobility by preemptively transferring user information to the next base station. They employed a Deep Q-Network (DQN) for content caching when such transferring is undergoing. Wu et al. [20] proposed a collaborative caching scheme using an asynchronous federated learning method to gather popular content information and determine the optimal collaborative caching strategy. He et al. [21] introduced a Multi-Agent Actor-Critic (MAAC) strategy empowered by a learning algorithm, where each Road Side Unit (RSU) decides its own caching schedules, enhancing caching efficiency. Somesula et al. [22] proposed a collaborative caching update method based on Multi-Agent Reinforcement Learning and Continuous Deep Deterministic Policy Gradient (MADDPG). Chen et al. [23] developed a collaborative edge caching algorithm that adopts an actor-critic framework for reducing data exchange between agents. Kong et al. [24] aimed to minimize the long-term energy consumption of a caching system by leveraging a DDPG for determining the schedules of computation offloading, service caching, and resource allocation.

3 System Models and Problem Formulation

3.1 System Model

In this paper, we consider an MEC environment with a central cloud server C_{cloud}, multiple base stations equipped with edge servers belonging to a set of $E = \{e_1, e_2, \cdots, e_i, \cdots, e_m\}$, multiple users $U = \{u_1, u_2, \cdots, u_j, \cdots, u_n\}$ and a set of content items $C = \{c_1, c_2, \cdots, c_k, \cdots, c_l\}$, as shown by Fig. 1.

The caching mechanism works according to a caching, transmission, and cost model. When a content request reaches the base station, it directly responds when the corresponding content is cached.

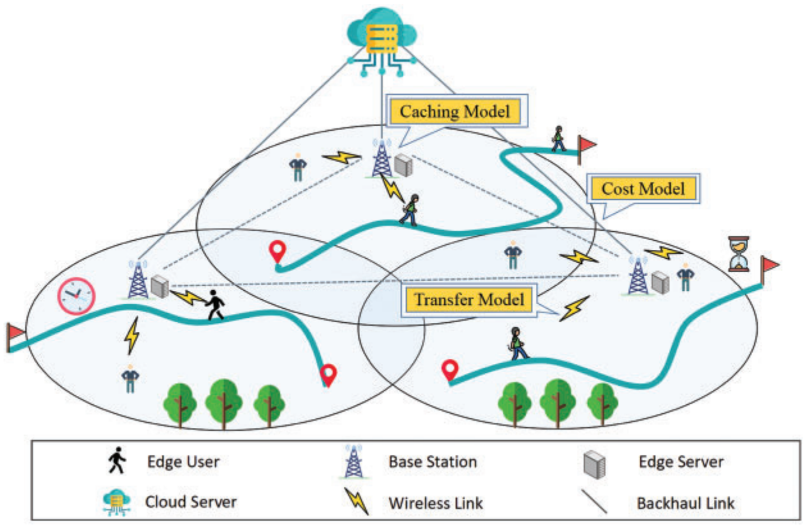

Figure 1: Edge computing system model

The central cloud server C_{cloud} is responsible for storing all content requested by users while the base stations E are distributed. Each base station can be described by a tuple $e_i = (LE_i, rad_i, cap_i, N_i)$, where LE_i represents the two-dimensional coordinates of base station e_i, rad_i the coverage radius of the base station, cap_i the current caching capacity of the base station and $N_i = \{e_{i1}, e_{i2}, \cdots, e_{iq}\}$ the set of neighboring base stations adjacent to base station e_i. This paper refers to Table 1 for annotations.

Table 1: Summary of the key notations

Notation	Definition
$E = \{e_1, e_2, \cdots, e_i, \cdots, e_m\}$	A set of base stations
N_i	A set of neighboring base station of the *i*-th base station
$cs_i(t)$	The caching state of the *i*-th base station at time t
cap_i	Capacity of the *i*-th base station
$U = \left\{u_1, u_2, \cdots, u_j, \cdots, u_n\right\}$	A set of users
$C = \{c_1, c_2, \cdots, c_k, \cdots, c_l\}$	A set of content
$csc_{i,k}(t)$	The cache status of the *k*-th content at the *i*-th base station
s_k	The size of content c_k
p_i	The transmission power for the *i*-th base station
$d_{i,j}(t)$	The request latency from the *j*-th user to the *i*-th base station at time *t*
$r_{i,j}(t)$	The transmission rate between the *j*-th user and the *i*-th base station at time *t*
$rd_i(t)$	The average request delay of the *i*-th base station at time *t*
$ur_{i,j}(t)$	A boolean indicator of whether the *j*-th user sends a request to the *i*-th base station at time *t*

(Continued)

Table 1 (continued)

Notation	Definition
$bc_{i,k}(t)$	A boolean indicator of whether the *i*-th base station caches the *k*-th content at time *t*
$cp_i(t)$	The cost of proactive caching at the *i*-th base station
$rad_i(t)$	The coverage radius of the *i*-th base station
$cpc_{i,k}(t)$	The cost of proactively caching the *k*-th content at the *i*-th base station at time *t*
$cr_{i,j}(t)$	The cost of the request from the *j*-th user to the *i*-th base station
$sc_i(t)$	The average system cost of the *i*-th base station
$WF(u_j)$	A set of weighted vectors of content interests for the *j*-th user
CIB_i	The set of content that is of interest to the *i*-th base station
$nb_i(t)$	The number of requests from users at the *i*-th base station at time *t*
$LU_j(t)$	The two-dimensional coordinates of *j*-th user at time *t*
LE_i	The two-dimensional coordinates of *i*-th base station
$L_i(t)$	The cache status of the predicted queue CIB_i at the *i*-th base station
$NE_i(t)$	The cache status of the predicted queue CIB_i in the neighboring base stations of the *i*-th base station
UV_j	The set of content vectors for the *j*-th user is generated by the Caser algorithm
CIU_j	The set of content that is of interest to the *j*-th user
$s_i(t), a_i(t), r_i(t)$	The set of state, action, reward of the *i*-th base station at time *t*
BCR	The basic cost of replacing content at the base station
BCH	The basic cost for handling user requests
W_t	The Caser global model parameters at time *t*
$H_i(t)$	The Caser local model parameters for the *i*-th base station at time *t*
rd_{max}, rd_{min}	The maximum and minimum values of the base station's average request delay.
sc_{max}, sc_{min}	The maximum and minimum values of the base station's average system cost

3.2 Storage Model

Given edge servers' limited storage capacity, it is impossible to cache all available content simultaneously. We use $csc_{i,k}(t)$ to indicate whether content c_k is cached by the base station e_i. Thus, the caching state of base station e_i can be represented as:

$$cs_i(t) = \left\{\mathrm{csc}_{i,1}(t),\ \mathrm{csc}_{i,2}(t), \cdots,\ \mathrm{csc}_{i,l}(t)\right\} \tag{1}$$

As the caching capacity of a base station is often limited by its total size:

$$\sum_{k=1}^{l} \left(csc_{i,k}(t) \cdot s_k\right) \le cap_i \tag{2}$$

where s_k represents the size of content c_k.

3.3 Transmission Model

At time t, user u_j requests content c_k from the nearest base station e_i. Upon receiving the request, base station e_i responds when the requested content is available or forwards the request to the neighboring base station e_{N_i} otherwise. When both stations fail, the request is subsequently sent to the cloud.

According to Shannon's formula [25], We have the delay of user u_j requesting content from e_i at time t as $d_{i,j}(t)$:

$$d_{i,j}(t) = \begin{cases} \frac{s_k}{r_{i,j}(t)}, & if \quad csc_{i,k}(t) = 1 \\ \frac{s_k}{r_{i,j}(t)} + \frac{s_k}{r_{N_i,i}(t)}, & if \quad csc_{i,k}(t) = 0 \ and \ \exists csc_{N_i,k}(t) = 1 \\ \frac{s_k}{r_{i,j}(t)} + \frac{s_k}{r_{C,i}(t)}, & \text{otherwise} \end{cases} \tag{3}$$

$$r_{i,j}(t) = b_i \cdot \log_2 \left(1 + \frac{p_i \cdot g_{i,j}}{\sigma^2}\right) \tag{4}$$

where $r_{i,j}(t)$ represents the transmission rate between base station e_i and user u_j. It depends on the transmission power p_i; channel gains $g_{i,j}$ (distance-dependent), Gaussian noise σ^2, and the bandwidth rate b_i.

Therefore, the average request delay for base station e_i is:

$$rd_i(t) = \frac{1}{nb_i(t)} \cdot \sum_{j=1}^{n} ur_{i,j}(t) \cdot d_{i,j}(t) \tag{5}$$

where $nb_i(t)$ represents the number of content requests to base station e_i at time t, and $ur_{i,j}(t)$ a boolean indicator of whether mobile device u_j requested content from e_i at time t.

3.4 Cost Model

The cost model comprises two main components: the cost of proactive caching and the cost of content delivery. In order to improve cache hit rates and minimize user request latency, base stations proactively cache a portion of popular content in the current server and replace unpopular content.

When base station e_i proactively caches content c_k at time t, we define the cost of proactive caching as $cpc_{i,k}(t)$:

$$cpc_{i,k}(t) = \begin{cases} 0, & if \quad csc_{i,k}(t) = 1 \\ p_{N_i} \cdot \frac{s_k}{r_{N_i,i}(t)} + BCR, & if \quad csc_{i,k}(t) = 0 \ and \ \exists csc_{N_i,k}(t) = 1 \\ p_C \cdot \frac{s_k}{r_{C,i}(t)} + BCR, & \text{otherwise} \end{cases} \tag{6}$$

where $csc_{i,k}(t)$ is a boolean indicator of whether the content actively cached by base station e_i is available in the current storage, and BCR represents the basic cost of replacing content at the base station.

The average cost of proactively caching content at the base station is:

$$cp_i(t) = \frac{1}{nb_i(t)} \cdot \left(\sum_{k=1}^{l} bc_{i,k}(t) \cdot cpc_{i,k}(t) \right) \tag{7}$$

where $bc_{i,k}(t)$ is a boolean indicator of whether the e_i caches c_k at time t.

The cost of the request, $cr_{i,j}(t)$, is decided by the energy required for user u_j to request content c_k:

$$cr_{i,j}(t) = \begin{cases} p_i \cdot \dfrac{s_k}{r_{i,j}(t)} + BCH, & if \quad csc_{i,k}(t) = 1 \\ p_i \cdot \dfrac{s_k}{r_{i,j}(t)} + p_{N_i} \cdot \dfrac{s_k}{r_{N_i,i}(t)} + BCH, & if \quad csc_{i,k}(t) = 0 \ and \ \exists csc_{N_i,k}(t) = 1 \\ p_i \cdot \dfrac{s_k}{r_{i,j}(t)} + p_C \cdot \dfrac{s_k}{r_{C,i}(t)} + BCH, & \text{otherwise} \end{cases} \tag{8}$$

where BCH represents the basic cost for handling user requests, and $BCR \gg BCH$.

The average system cost of base station e_i is:

$$sc_i(t) = \frac{1}{nb_i(t)} \cdot \left(\sum_{j=1}^{n} ur_{i,j}(t) \cdot cr_{i,j}(t) + \sum_{k=1}^{l} bc_{i,k}(t) \cdot cpc_{i,k}(t) \right) \tag{9}$$

3.5 Problem Formulation

As mentioned earlier, latency is a leading factor in deciding the effectiveness of the caching system. Nevertheless, the goal of reducing latency can usually conflict with other goals, e.g., reducing cost [26]. The objective of this work is thus to reconcile conflicting goals:

$$\text{Min: } \frac{1}{m} \cdot \sum_{t=0}^{t_{max}} \sum_{i=0}^{m} \left[w_1 \cdot \frac{rd_i(t) - rd_{min}}{rd_{max} - rd_{min}} + w_2 \cdot \frac{sc_i(t) - sc_{min}}{sc_{max} - sc_{min}} \right] \tag{10}$$

$$\begin{aligned} s.t. \quad & \textbf{C1.} \quad \textstyle\sum_{i=0}^{m} ur_{i,j}(t) = 1 && \forall j, \forall t \\ & \textbf{C2.} \sqrt{(LU_j(t) - LE_i)^2} < rad_i && \forall i, \forall j, \forall t \\ & \textbf{C3.} \textstyle\sum_{k=1}^{l} \left(csc_{i,k}(t) \cdot s_k \right) \le cap_i && \forall j, \forall t \end{aligned}$$

where w_1 and w_2 are the weighting factors for latency and cost, respectively. $LU_j(t)$ denotes the two-dimensional coordinates of mobile device u_j at time t. rd_{max}, rd_{min}, and sc_{max}, sc_{min} represent the maximum and minimum values of the base station's average request delay and system cost, respectively. C1 ensures users request content from only one base station at a time, maintaining network efficiency. C2 aims to ensure that the distance between users and the base station is within the base station's coverage radius to ensure reliable connectivity. C3 indicates that all content cached on the base station does not exceed its storage capacity, ensuring it remains within storage capacity limits to prevent performance degradation. Clearly, the proposed optimization problem falls under the Mixed-Integer Nonlinear Programming (MINLP) category, known for its computational challenges and recognition as NP-hard. MINLP uniquely integrates continuous and discrete variables within nonlinear constraints, reflecting the nuanced demands of real-world problems. The sophisticated nature of MINLP means that finding an exact solution within a reasonable timeframe is often impractical. This reality, especially apparent in our model's blend of continuous variables (e.g., device coordinates) and discrete decisions (e.g., choices about content caching) alongside nonlinear constraints (e.g., ensuring connectivity within a base station's coverage radius), necessitates the pursuit of heuristic or approximate methods. Our approach aims to provide a practical solution to the challenges posed by this complex problem in edge caching.

4 The Proposed Method

This chapter provides a comprehensive introduction to the PRIME model, which utilizes content popularity prediction for forecasting the popularity ranking CIB_i of content at the base station e_i. The PRIME model incorporates the Caser algorithm for content popularity prediction. In doing so, the stay time of users is used as a weighting factor for content, and the algorithm increases the likelihood of caching content that is appealing to users with a long stay time. To avoid the imbalance of the model, the parameters of the Caser algorithm are periodically updated for each base station. The computed popularity rankings CIB_i are fed into the reinforcement learning-based content decision model. This model takes into account the caching status of neighboring and current base stations, as well as the request counts, aiming to minimize both the system cost and the request delay. It is important to emphasize that the algorithm in this paper assumes that users send requests to the neighboring base stations at fixed time intervals, and each user's position remains constant during each interval. This assumption aids in building the model to address content caching issues and provides performance optimization solutions in practical edge computing environments [27]. Fig. 2 presents the architecture of the PRIME model.

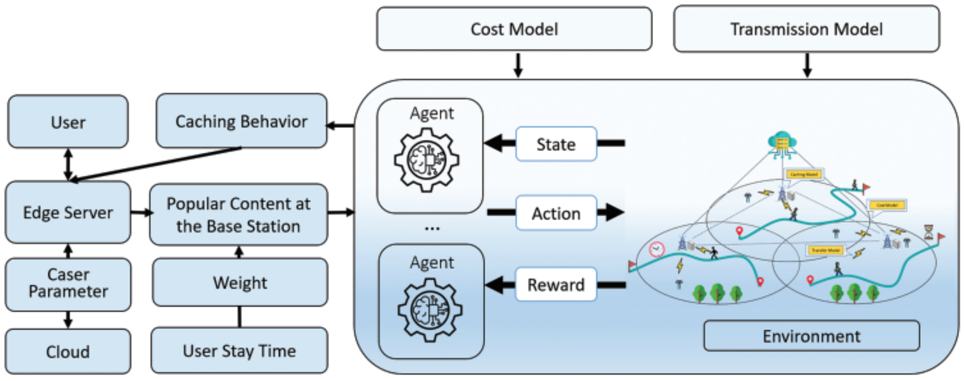

Figure 2: Framework of PRIME for yielding the caching strategy

4.1 Content Popularity Prediction Model

The popularity prediction model takes into account users' stay time and their request traces, as illustrated in Fig. 3. The model comprises four sequential steps:

Global Model Download: At the beginning of each time t, every base station retrieves the Caser global model parameters denoted as W_t from the cloud. The Caser model's fundamental function is to predict the content that users may be interested in the near future based on their recent request traces. This enables each base station to accurately identify the content of interest for each user, thus providing robust support for content caching decisions.

Acquiring User-Interest Content: Upon receiving the Caser global model parameters W_t, base station e_i utilizes the most recent user request trace for local Caser model iterative updates (Lines 4–6). The Caser algorithm predicts user behaviors in recommendation systems by embedding user-item interactions and leverages convolutional neural networks for identifying temporal patterns. Once the update is complete, the updated local Caser model $H_i(t)$ is uploaded to the cloud (Line 8). In the training process, the Caser model extracts a request data sequence of length g from the user's recent request trace. The first p times' request contents serve as Caser's input, while the remaining h times' request contents are treated as predictive ($h = g - p$). Caser's processing steps involve extracting features of users and requested content through an embedding layer, followed by applying horizontal and vertical convolutions to the requested content to capture diverse interest patterns within the user's historical behavior sequence. Horizontal convolution focuses on long-term interest variations, while vertical convolution is better suited for modeling users' short-term interests. Finally, more advanced and abstract features are generated through a fully connected neural network layer that integrates the outputs from horizontal and vertical convolutions and user characteristics, thereby producing the weights of user requests for each content item. During the prediction phase, the algorithm generates request weights for each content for users based on the user's most recent p content requests. The top k_1 contents with the highest weights are the ones that u_j is interested in, denoted as CIU_j (Line 10). Caser employs a binary cross-entropy loss function, where smaller losses are associated with higher request weights for predicted content and lower request weights for non-predicted content. The loss function for user u_j is:

$$\text{loss}_{u_j} = \sum_{t\in h}\sum_{k\in ct_j(t)}\left[-\log(sig(\boldsymbol{y}_{j,k}(t))) + \sum_{w\neq k} -\log(1 - sig(\boldsymbol{y}_{j,k}(t)))\right] \tag{11}$$

where $ct_j(t)$ represents the content requested by user u_j within the h times, $\boldsymbol{y}_{j,k}(t)$ denotes Caser's request weight for content c_k, sig represents the sigmoid function, and $w \neq k$ signifies that content c_w was not requested within the h times.

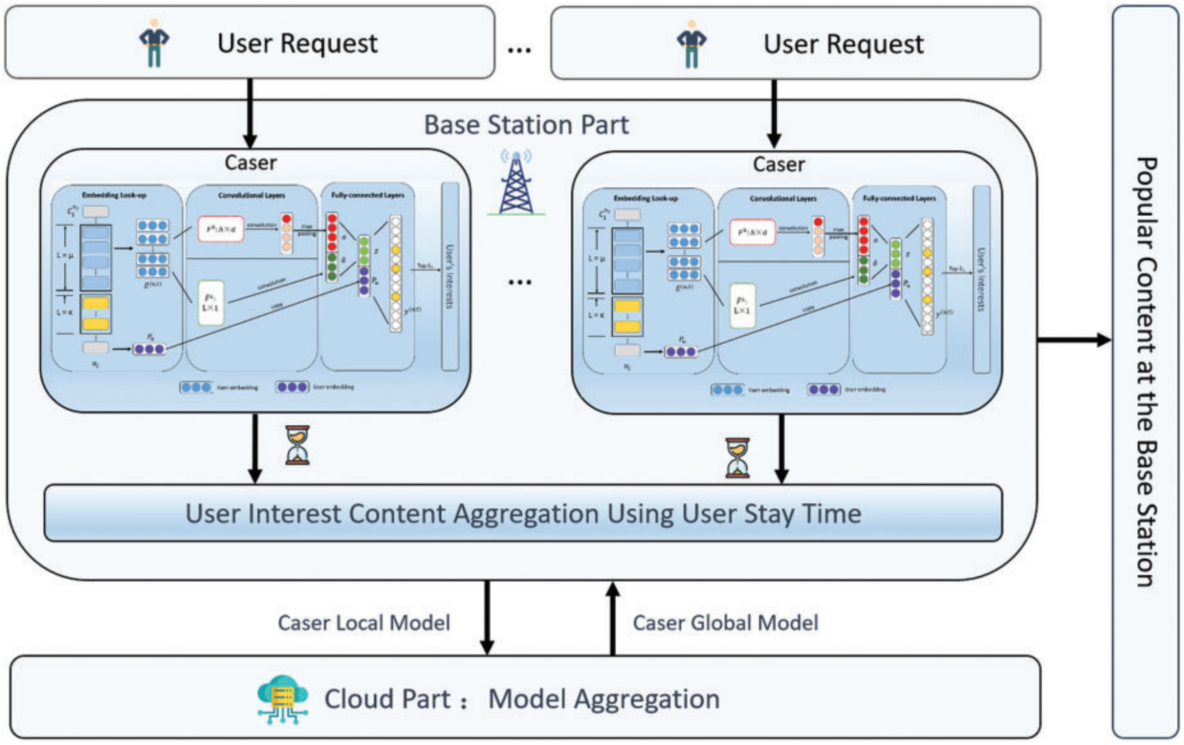

Figure 3: Framework of PRIME for yielding the caching strategy

Retrieving Currently Popular Content at the Base Stations: Once the base station obtains the content of interest for all users within its coverage, it weighs these contents to determine the most popular content (Line 11). In this process, the base station employs the stay time of users as an aggregation weight for content, where content requested by users with higher weight is more likely to be cached, according to the stay times of users. This mechanism helps avoid caching content requested by users with short stay times, thus bringing in a high overhead of cache replacement. The stay time of users is estimated according to the coverage radius of the corresponding base station and its mobile trajectory, which falls into such coverage area. The weight is thus:

$$WF(u_j) = \begin{cases} \frac{t_{stay}}{2t_{con}} UV_j\, (1 + sgn(t_{stay} - th)) & (t_{stay} < t_{con}) \\ UV_j & otherwise \end{cases} \tag{12}$$

where t_{con} is the maximum time allowed for predicting user preferences in the Caser algorithm, $UV_j = \{ci_1, ci_2, \cdots, ci_l\}$ a set of content vectors generated by the Caser algorithm, ci_k a boolean indicator of whether content c_k attracts user u_j, sgn a sign function [28] and th a threshold value for excluding users with stay times lower than it.

Each base station maintains the weights $WF(u_j)$ and selects the top k_2 contents of interest CIB_i (Line 14).

$$CIB_i = \max_{p=1}^{k_2} \sum_{j=0}^{n} ur_{i,j}(t) \cdot WF(u_j) \tag{13}$$

Model Aggregation: Upon receiving the local models H_t uploaded from each base station, the cloud server updates the global model W_{t+1} (Line 16). To address the issue of imbalanced training data in local models of different base stations, the algorithm assigns varying weights to the local models uploaded from different base stations for aggregation. In this scenario, the updated global model W_{t+1} is:

$$\mathbf{W}_{t+1} = \mathbf{W}_t + \sum_{e_i \in E} \frac{nb_i(t)}{\sum_{e_i \in E} nb_i(t)} (\mathbf{H}_i(t) - \mathbf{W}_t) \tag{14}$$

where $nb_i(t)$ represents the number of user content requests received by base station e_i at time t.

4.2 Deep Reinforcement Learning for Cache Decision Making

Increasing caching capacity helps improve the user-experienced QoS in terms of cache hit rate but can increase caching cost and energy. Due to capacity limitations, not all popular content can be cached at the base station [29]. It is thus clear that the objectives of minimizing system cost and improving user QoS are conflicting and should be reconciled. To address this challenge, we leverage a DQN, a type of deep reinforcement learning model, to yield high-quality caching plans according to the optimization formulation given in Eq. (10). This approach allows for dynamic adjustment of caching strategies based on learning from user demands and system constraints, as illustrated in Fig. 4. The model comprises the following components:

Algorithm 1: Predicting popularity using stay time

Input: A set of user request traces and mobile trajectories.
Output: The predicted popularity priority queue CIB.

```
1:  for each $t \in T$ do
```

(Continued)

Algorithm 1 (continued)

```
2:      for each $e_i \in E$ do
3:          Download the Caser global model $W_t$
4:          for each user's recent request traces cached on $e_i$ do
5:              Calculate the predicted loss according to Eq. (11).
6:          end for
7:          Update model parameters $H_i(t)$
8:          Upload $H_i(t)$ to the CS
9:          for each $u_j$ requesting content from $e_i$ at time t do
10:             Obtain the top $k_1$ content according to Caser.
11:             Calculate the weighted $WF(u_j)$ according to Eq. (12).
12:             Update recent request traces for $u_j$
13:         end for
14:         Obtain the predicted queue $CIB_i$ under $e_i$ according to Eq. (13).
15:     end for
16:     The CS update $W_{t+1}$ according to Eq. (14).
17: end for
18: return The predicted popularity priority queue $CIB$.
```

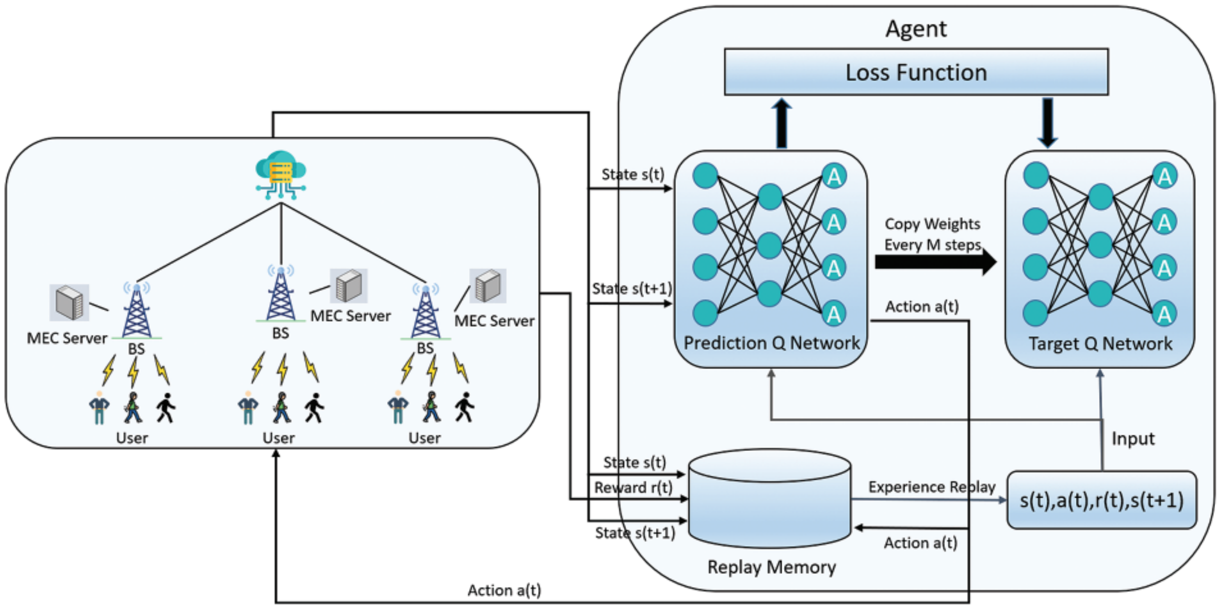

Figure 4: Framework of PRIME for yielding the caching strategy

State: The state space $s_i(t)$ of base station e_i involves the cache status of the predicted queue CIB_i in e_i, the cache status of CIB_i in the neighboring base stations $NE_i(t)$ and the number of user requests at e_i for the next time $m_i(t+1)$. Therefore, the state can be described as a tuple $s_i(t) = (L_i(t), NE_i(t), m_i(t+1))$, where $L_i(t) = \{L_{i,1}, L_{i,2}, \cdots, L_{i,k_2}\}$, $L_{i,k}$ represents whether the k-th popular content is cached in e_i, $NE_i(t) = \{NE_{i,1}, NE_{i,2}, \cdots, NE_{i,k_2}\}$ and $NE_{i,k}$ whether the k-th popular content is cached in the neighboring base stations. $m_i(t+1)$ can be estimated based on the coverage radius of the corresponding base station and the dwell time of users who stay within the current area.

Actions: We define actions as $a = (a_1, a_2, \cdots, a_n)$, where $a_i = (a_{i,1}, a_{i,2}, \cdots, a_{i,k_2})$ represents the caching decision for popular content at base station e_i. In this context, $a_{i,k}$ is a boolean indicator of whether caching the k-th popular content is necessary. While $a_{i,k} = 1$ denotes that it is necessary to cache the k-th popular content. To cache the necessary content, the base station removes less popular content. The action space size can be estimated in the following way: each base station has the ability to cache multiple contents. However, the DQN model outputs only a single action per decision. With a total of k_2 candidate contents, the total size of the action space is the sum of combinations from 0 to k_2, which is $C_{k_2}^0 + C_{k_2}^1 + C_{k_2}^2 + \cdots + C_{k_2}^{k_2}$. To simplify, we divide the contents into v segments (where $v \ll k_2$), which significantly reduces the action space to $C_{k_2}^0 + C_v^1 + C_v^2 + \cdots + C_v^v$. The choice of action function follows the ε-greedy method to balance exploration and exploitation for optimizing content caching strategies. The calculation formula is:

$$a(t) = \underset{a}{argmax}(Q(s(t), a; \boldsymbol{\theta})) \tag{15}$$

Reward: It aims to minimize the cost for each base station with the constraints of user QoS. The reward function is:

$$r_i(t) = w_s e^{-scn_i(t)} + w_r e^{-rdn_i(t)} \tag{16}$$

where $scn_i(t)$ represents the normalized cost of $sc_i(t)$ at the base station e_i and $rdn_i(t)$ the normalized value of the request delay $rd_i(t)$. w_s and w_r are the weighting factors for cost and request delay.

We employed the DQN reinforcement learning algorithm to dynamically adjust cache content based on users' historical data and behavior, aiming to optimize system performance. The DQN algorithm, an amalgamation of deep learning and reinforcement learning, utilizes deep neural networks to approximate the Q-function, denoted as $Q(s_i, a_i; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ represents the reinforcement learning parameters, this $Q(s_i, a_i; \boldsymbol{\theta})$ is also known as the prediction network. This represents the Q-value when the agent is in the state s_i and takes action a_i, enabling DQN to handle high-dimensional sensory inputs effectively. Key features of the DQN algorithm include experience replay and a target network. Experience replay involves storing the agent's experiences in a memory pool and randomly sampling these experiences for network training, reducing correlation between samples and enhancing learning stability. The target network, denoted as $\hat{Q}\left(s_i, a_i; \boldsymbol{\theta}'\right)$, where $\boldsymbol{\theta}'$ represents the parameters of the target network, serves as a periodically updated replica of the Q-network. It is used for calculating the target Q-values and is crucial in enhancing the algorithm's performance and stability.

In our study, within the DQN framework, each base station functions as an agent. At each base station e_i, the process begins by taking an action $a_i(t)$ based on the current state $s_i(t)$ to decide which content to cache. Following this action, the base station receives a $r_i(t)$ reward and transitions to the next state $s_i(t+1)$. This progression forms a state transition, denoted as $(s_i(t), a_i(t), r_i(t), s_i(t+1))$, which is then stored in a replay buffer [30]. Subsequently, P tuples are randomly selected from the replay buffer to create a minibatch, with the p-th tuple represented as (s_p, a_p, r_p, s_{p+1}). The loss function used for updating parameters is:

$$L(\boldsymbol{\theta}) = \frac{1}{P}\sum_{p=1}^{P}\left[\left(y_p - Q\left(s_p, a_p; \boldsymbol{\theta}\right)\right)^2\right] \tag{17}$$

where y_p represents the target Q-value of the target network, and the target network's parameters $\boldsymbol{\theta}'$ are periodically updated to match the prediction target network's parameters $\boldsymbol{\theta}$ (Lines 18–19), y_p is calculated as:

$$y_p = r_p + \gamma \max_a \hat{Q}\left(s_{p+1}, a; \boldsymbol{\theta}'\right) \tag{18}$$

where γ represents the discount factor, $\gamma \in (0, 1)$. The gradient of the loss function for all elements taken from the replay buffer is calculated as:

$$\nabla_\theta L(\boldsymbol{\theta}) = \frac{1}{P}\sum_{p=1}^{P}\left[\left(y_p - Q\left(s_p, a_p, \boldsymbol{\theta}\right)\right)\nabla_{\boldsymbol{\theta}^p} Q\left(s_p, a_p, \boldsymbol{\theta}\right)\right] \tag{19}$$

At the end of time t, the prediction network's parameters $\boldsymbol{\theta}$ are updated:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta_\theta \nabla_\theta L(\boldsymbol{\theta}) \tag{20}$$

where η_θ is the learning rate for the prediction network.

Algorithm 2: Predicting popularity using stay time

Input: A set of content requests and the predicted popularity priority queue *CIB*.
Output: The caching decisions.

```
1:  for each $t = 1, 2 \cdots, T$ do
2:    for $i = 1, 2 \cdots, m$ do
3:      Obtain the predicted popularity priority queue $CIB_i$ according to Algorithm 1
4:      Obtain the state s(t) according to environment
5:      Calculate the Q-value of the target network according to Eq. (18).
6:      Calculate the action a(t) according to Eq. (15).
7:      Obtain the next state s(t + 1) after executing a(t).
8:      Obtain cost $sc_i(t)$ according to Eq. (9).
9:      Obtain latency $rd_i(t)$ according to Eq. (5).
10:     Obtain the reward r(t) according to Eq. (16).
11:     Store the tuple (s(t), a(t), r(t), s(t + 1)) and randomly sample a minibatch from it.
12:     Calculate the loss function by Eq. (17).
13:     Calculate the gradient by Eq. (19).
14:     Update parameters $\theta$ according to Eq. (20).
15:   end for
16:   Obtain the caching decisions according to $\theta$.
17:   Each base station selects contents for replacement from the prediction queue $CIB_i$ based on
      caching decisions.
18:     if the number of slots is $z$ then
19:       $\boldsymbol{\theta}' = \boldsymbol{\theta}$
20:     end if
21:   end for
22:   return The caching decisions
```

At each time step t, **Algorithm 1** is used to generate a predicted popularity priority queue Q_{k_2}. This algorithm integrates the priority queue with environmental factors, including the base station's current caching status, neighboring base stations' caching status, and the number of user requests, to formulate the base station's state $s(t)$ (Lines 3–4). Following this, a deep reinforcement learning model is employed. This model leverages the cost function from Eq. (9) and the delay function from Eq. (5) as its reward functions, aiming to optimize cache decision making (Lines 5–14). Ultimately, based on the results of this training, each base station selects content from the predictive queue Q_{i,k_2} for caching or replacement (Lines 16–17).

5 Performance Evaluation

5.1 Simulation Configuration

In this experiment, we employed two datasets: MovieLens 1M [31] and Shanghai Telecom [32]. The Shanghai Telecom dataset was used to analyze user mobility, which features over 7.2 million content access event records from 9,481 mobile users and 3,233 edge base stations. This dataset also includes detailed mobile trajectory information, as shown in Fig. 5. For training and testing user request patterns for interesting content, we utilized the MovieLens 1M dataset, which comprises around one million ratings from 6,040 anonymous users.

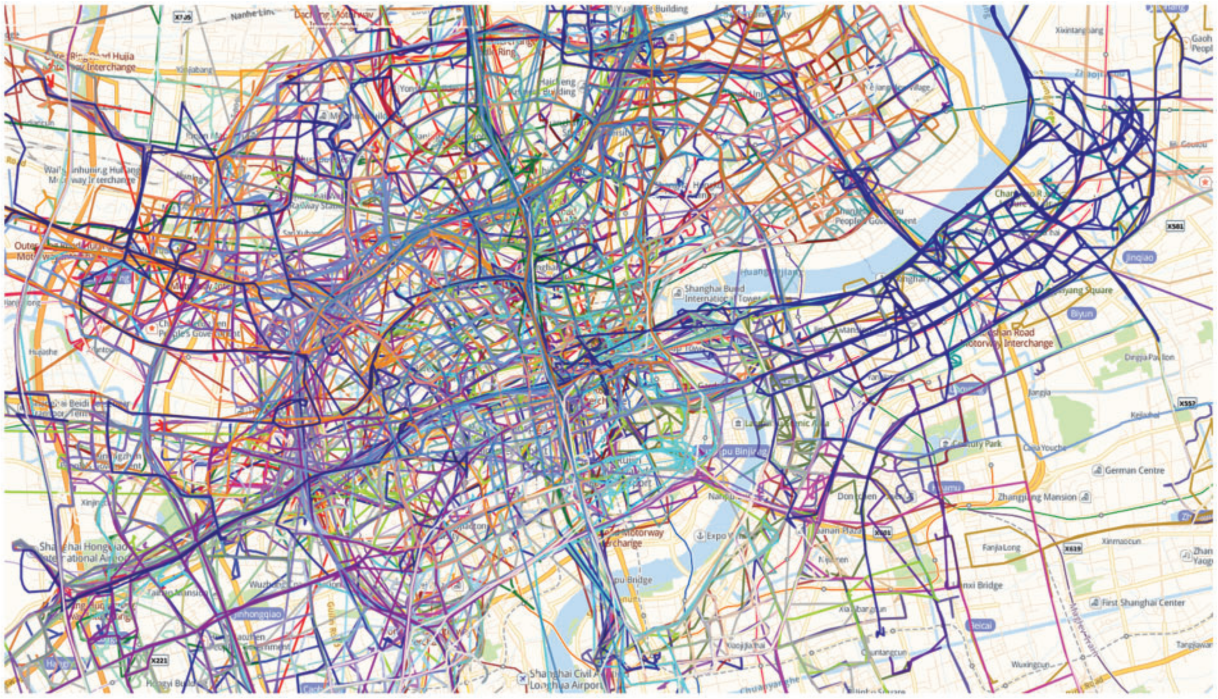

Figure 5: Sample user trajectories of Shanghai Telecom

To simulate user content request processes, we regarded user movie ratings as requests for the corresponding movies [33]. The distribution of users and edge servers is shown in Fig. 6. We selected 3,350 users with at least 80 ratings from MovieLens and paired them with corresponding trajectories from the Shanghai Telecom dataset. The key criterion for this pairing was ensuring that the trajectories in the study area were longer than the users' rating records, thereby aligning user requests with the study area. The Caser algorithm was pre-trained using the remaining data. Users send requests to the nearest base station when they are within the coverage of multiple edge servers. Other parameters are valued according to [20,34], and are given in Table 2.

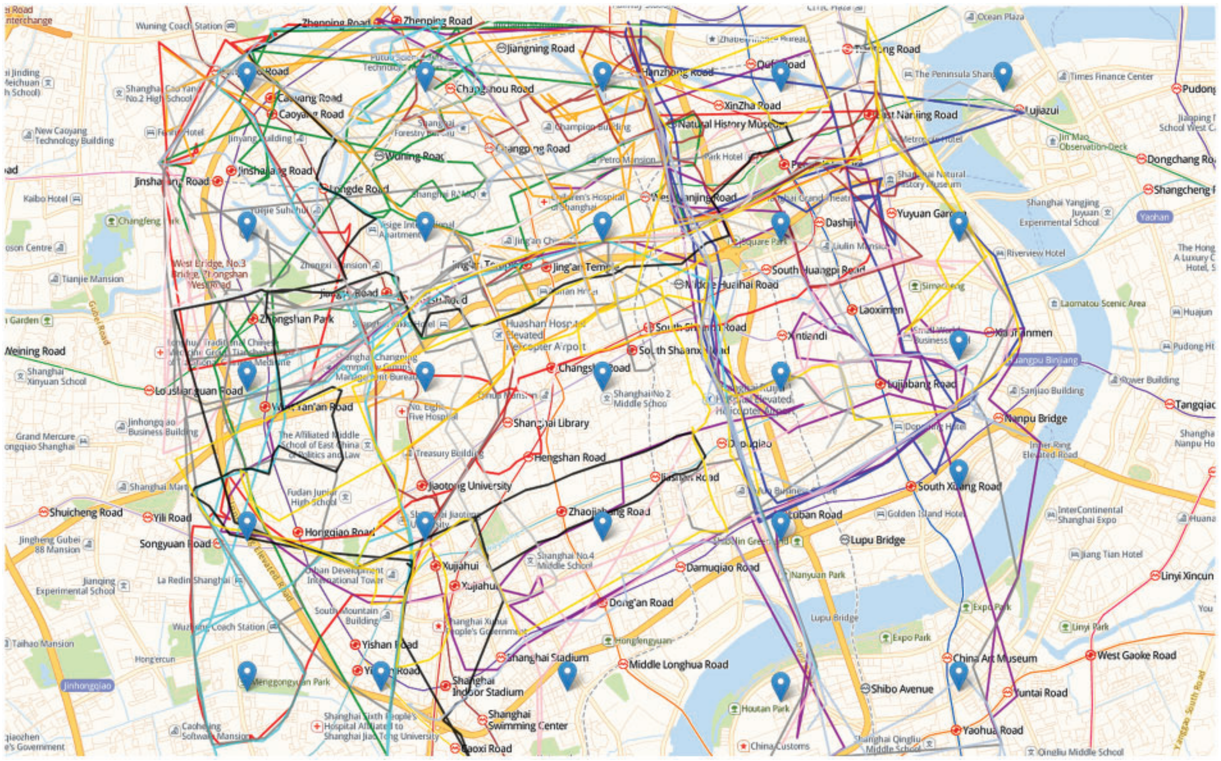

Figure 6: Partially selected user trajectory and base station deployment locations

Table 2: Parameter table

Parameter	Value
Number of users	3350
Edge server capacity in the base station	0–300
Number of contents that attract users	4–6
Coverage radius (m)	1250
Bandwidth between base stations and users (MHz)	20
Bandwidth between base stations (MHz)	50
Bandwidth between base stations and cloud (MHz)	200
Learning rate for DRL	0.001
Reward decay factor for DRL	0.95
Total rounds of simulation	300

5.2 Baseline Algorithms

We compare our method against four baselines:

1) Baseline Algorithm 1 (BA1) [19]: A wireless edge caching method based on deep reinforcement learning. This method determines content to be cached according to the uneven distribution of file popularity and user mobility.

2) Baseline Algorithm 2 (BA2) [9]: A cooperative caching method that utilizes LSTM networks for predicting content popularity. It leverages the content size-based weights for trading off content popularity and size's impact. It takes the mobility of users as inputs for yielding caching schedules.

3) First-In-First-Out Scheme (FIFO): Base stations cache content in the order of content requests and discard the earliest cached content when the cache space is exhausted.

4) Random: Base stations randomly cache a portion of the content.

5.3 Performance Analysis

Fig. 7 compares cache hit rates of different methods with varying cache capacities. As the edge server capacity increases, base stations can cache more content, thereby enhancing the opportunities for users to access resources from local edge servers and neighboring edge servers. Consequently, the hit rates increase with the capacity. PRIME outperforms FIFO, BA1, and BA2 strategies in terms of edge cache hit rate by 45.11%, 15.69%, and 9.08% on average, respectively.

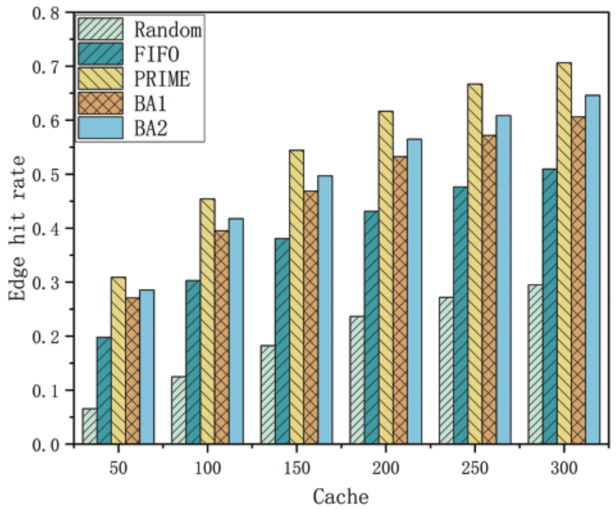

Figure 7: Capacity and edge hit rate

Fig. 8 displays the average request latency for different cache strategies with varying cache capacities. It is clear that PRIME achieves lower latency than its peers. As the cache capacity of the edge server increases, the request latency decreases for all cache strategies. This is because a larger cache capacity increases the likelihood of mobile users accessing content from local and nearby servers, thereby reducing the request latency. PRIME outperforms Random, FIFO, BA1, and BA2 by 30.24%, 17.28%, 8.43%, and 5.31%, respectively.

Fig. 9 shows the average cost of proactive caching for different methods at varying cache capacities. As cache capacity increases, the cost of proactive caching also rises. This increase is attributed to the server's enhanced ability to store more content, which reduces user latency but increases caching costs. Furthermore, its growth tends to decrease with capacity because higher hit rates help to alleviate the cost of communication with the remote cloud. It is important to note that since FIFO recommends only one content per user at a time, when the cache capacity reaches 150, the content that needs to be proactively cached by the base station no longer increases. In contrast, PRIME saves proactive caching cost by 72.59%, 36.71%, 23.19%, and 17.41% in comparison with Random, LRU, BA1 and BA2, respectively.

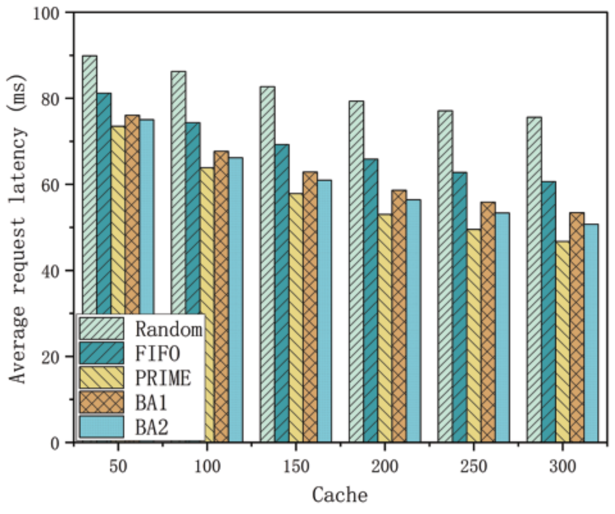

Figure 8: Capacity and average request latency

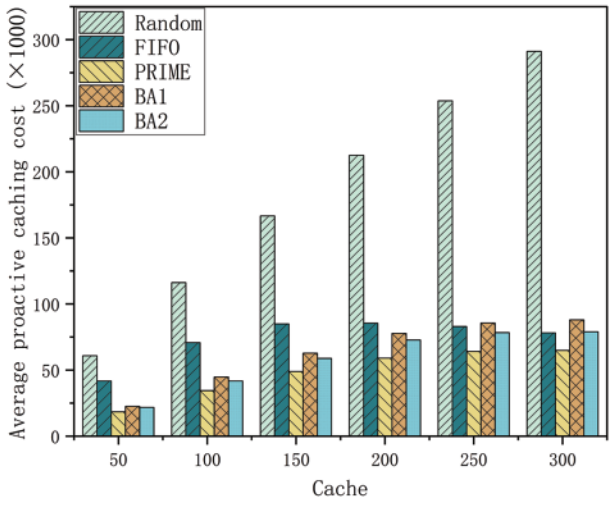

Figure 9: Capacity and average cost of proactive caching

Fig. 10 displays the system's average cost with different methods and varying cache capacities. As is evident from all methods, the system's cost comprises two primary components: the cost of proactive caching at the base stations and the cost of user requests. As server cache capacity increases, the proactive cache cost at base stations gradually rises. However, the cost of user requests decreases due to the gradual reduction in user request latency. Thus, the system's average cost does not necessarily increase with the increased capacity. PRIME demonstrates a lower system cost than FIFO, BA1, and BA2 by 54.85%, 26.02%, 15.31%, and 10.62%, respectively.

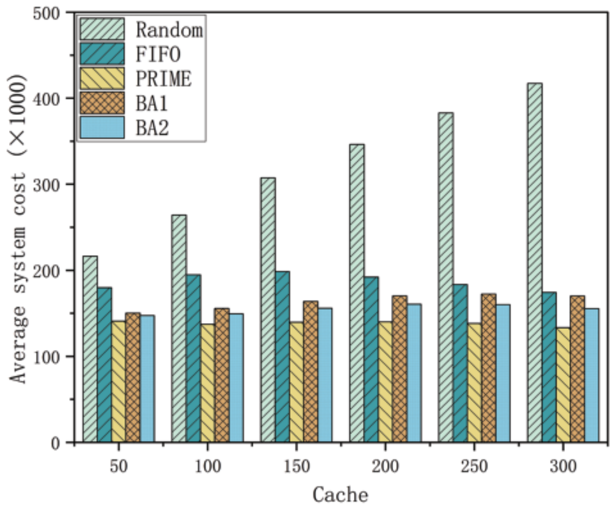

Figure 10: Capacity and average system cost

6 Conclusion

In this paper, we introduced PRIME, a predictive user popularity-aware approach for proactive content caching in MEC. PRIME synthesizes a content popularity prediction model and a deep reinforcement learning procedure for yielding dynamic caching schedules. It generates accurate and evolving popularity score estimates for cached content, leading to superior predictive and proactive caching plans. Our experiments show that PRIME surpasses existing methods in various performance metrics.

In the future, we plan to incorporate anomaly detection models to refine content caching strategies and deepen our analysis of user behavior. This will involve examining temporal patterns in user requests and mobility to enhance PRIME's predictive precision. Furthermore, we aim to explore PRIME's scalability in more extensive and complex network settings and its adaptability to different user densities and mobility scenarios. Integrating edge computing with emerging technologies like 5G and IoT devices is another exciting direction to improve MEC content caching efficiency. These focused efforts will progressively advance proactive caching in MEC environments.

Acknowledgement: The authors wish to express their gratitude to the reviewers for their insightful suggestions, which have significantly enhanced the clarity and presentation of this paper, and extend their thanks to the editors for their invaluable support and guidance.

Funding Statement: The authors received no specific funding for this study.

Author Contributions: The authors confirm contribution to the paper as follows: study conception and design: Yunni Xia; data collection: Yunye Wan, Peng Chen; analysis and interpretation of results: Yunye Wan, Yong Ma, Dongge Zhu; draft manuscript preparation: Yunye Wan, Xu Wang, Hui Liu; manuscript review and editing: Yunni Xia, Weiling Li, Xianhua Niu, Lei Xu, Yumin Dong. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The datasets used in this study are publicly available. The MovieLens dataset is detailed in [31] and can be accessed as described therein. The Shanghai Telecom dataset is elaborated in [32], with acquisition information provided in the respective publication. Both datasets have been utilized by their respective usage guidelines and terms of service.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

1. Sahoo, S., Mukherjee, A., Halder, R. (2021). A unified blockchain-based platform for global e-waste management. *International Journal of Web Information Systems, 17(5),* 449–479. https://doi.org/10.1108/IJWIS-03-2021-0024
2. Liu, J., Wu, Z., Liu, J., Zou, Y. (2022). Cost research of Internet of Things service architecture for random mobile users based on edge computing. *International Journal of Web Information Systems, 18(4),* 217–235. https://doi.org/10.1108/IJWIS-02-2022-0039
3. Gao, H., Yu, X., Xu, Y., Kim, J. Y., Wang, Y. (2024). MonoLI: Precise monocular 3D object detection for next-generation consumer electronics for autonomous electric vehicles. *IEEE Transactions on Consumer Electronics,* 1.
4. Uthansakul, P., Khan, A. A. (2019). On the energy efficiency of millimeter wave massive MIMO based on hybrid architecture. *Energies, 12(11),* 2227. https://doi.org/10.3390/en12112227
5. Ioannou, A., Weber, S. (2016). A survey of caching policies and forwarding mechanisms in information-centric networking. *IEEE Communications Surveys & Tutorials, 18(4),* 2847–2886. https://doi.org/10.1109/COMST.2016.2565541
6. Ahlehagh, H., Dey, S. (2012). Video caching in radio access network: Impact on delay and capacity. *2012 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 2276–2281. Paris, France.
7. Shuja, J., Bilal, K., Alasmary, W., Sinky, H., Alanazi, E. (2021). Applying machine learning techniques for caching in next-generation edge networks: A comprehensive survey. *Journal of Network and Computer Applications, 181,* 103005. https://doi.org/10.1016/j.jnca.2021.103005
8. Garg, N., Sellathurai, M., Bhatia, V., Bharath, B. N., Ratnarajah, T. (2019). Online content popularity prediction and learning in wireless edge caching. *IEEE Transactions on Communications, 68(2),* 1087–1100.
9. Li, L., Kwong, C. F., Liu, Q., Kar, P., Ardakani, S. P. (2021). A novel cooperative cache policy for wireless networks. *Wireless Communications and Mobile Computing, 2021,* 1–18.
10. Yu, Z., Hu, J., Min, G., Zhao, Z., Miao, W. et al. (2020). Mobility-aware proactive edge caching for connected vehicles using federated learning. *IEEE Transactions on Intelligent Transportation Systems, 22(8),* 5341–5351.
11. Qi, K., Yang, C. (2020). Popularity prediction with federated learning for proactive caching at wireless edge. *IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1–6. Seoul, Korea (South).
12. Zhang, Y., Li, Y., Wang, R., Lu, J., Ma, X. et al. (2020). PSAC: Proactive sequence-aware content caching via deep learning at the network edge. *IEEE Transactions on Network Science and Engineering, 7(4),* 2145–2154. https://doi.org/10.1109/TNSE.6488902
13. Gao, H., Qiu, B., Wang, Y., Yu, S., Xu, Y. et al. (2023). TBDB: Token bucket-based dynamic batching for resource scheduling supporting neural network inference in intelligent consumer electronics. *IEEE Transactions on Consumer Electronics, 1.*
14. Uthansakul, P., Khan, A. A. (2019). Enhancing the energy efficiency of mmWave massive MIMO by modifying the RF circuit configuration. *Energies, 12(22),* 4356. https://doi.org/10.3390/en12224356
15. Wei, X., Liu, J., Wang, Y., Tang, C., Hu, Y. (2021). Wireless edge caching based on content similarity in dynamic environments. *Journal of Systems Architecture, 115,* 102000. https://doi.org/10.1016/j.sysarc.2021.102000

16. Gao, H., Wu, Y., Xu, Y., Li, R., Jiang, Z. (2023). Neural collaborative learning for user preference discovery from biased behavior sequences. *IEEE Transactions on Computational Social Systems,* 1–11.
17. Tang, J., Wang, K. (2018). Personalized top-N sequential recommendation via convolutional sequence embedding. *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pp. 565–573. New York, USA.
18. Zhu, H., Cao, Y., Wang, W., Jiang, T., Jin, S. (2018). Deep reinforcement learning for mobile edge caching: Review, new features, and open issues. *IEEE Network, 32(6),* 50–57. https://doi.org/10.1109/MNET.2018.1800109
19. Cai, Y., Chen, Y., Ding, M., Cheng, P., Li, J. (2021). Mobility prediction-based wireless edge caching using deep reinforcement learning. *2021 IEEE/CIC International Conference on Communications in China*, pp. 1036–1041. Xiamen, China.
20. Wu, Q., Zhao, Y., Fan, Q., Fan, P., Wang, J. et al. (2022). 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.
21. He, P., Cao, L., Cui, Y., Wang, R., Wu, D. (2023). Multi-agent caching strategy for spatial-temporal popularity in IoV. *IEEE Transactions on Vehicular Technology, 72(10),* 13536–13546. https://doi.org/10.1109/TVT.2023.3277191
22. Somesula, M. K., Rout, R. R., Somayajulu, D. V. (2022). Cooperative cache update using multi-agent recurrent deep reinforcement learning for mobile edge networks. *Computer Networks, 209,* 108876. https://doi.org/10.1016/j.comnet.2022.108876
23. Chen, S., Yao, Z., Jiang, X., Yang, J., Hanzo, L. (2020). Multi-agent deep reinforcement learning-based cooperative edge caching for ultra-dense next-generation networks. *IEEE Transactions on Communications, 69(4),* 2441–2456.
24. Kong, X., Duan, G., Hou, M., Shen, G., Wang, H. et al. (2022). Deep reinforcement learning-based energy-efficient edge computing for internet of vehicles. *IEEE Transactions on Industrial Informatics, 18(9),* 6308–6316. https://doi.org/10.1109/TII.2022.3155162
25. Gao, H., Wang, X., Wei, W., Al-Dulaimi, A., Xu, Y. (2024). Com-DDPG: Task offloading based on multiagent reinforcement learning for information-communication-enhanced mobile edge computing in the internet of vehicles. *IEEE Transactions on Vehicular Technology, 73(1),* 348–361. https://doi.org/10.1109/TVT.2023.3309321
26. Jiang, W., Feng, D., Sun, Y., Feng, G., Wang, Z. et al. (2021). Proactive content caching based on actor-critic reinforcement learning for mobile edge networks. *IEEE Transactions on Cognitive Communications and Networking, 8(2),* 1239–1252.
27. Zulfa, M. I., Hartanto, R., Permanasari, A. E. (2020). Caching strategy for Web application-a systematic literature review. *International Journal of Web Information Systems, 16(5),* 545–569. https://doi.org/10.1108/IJWIS-06-2020-0032
28. Zhang, R., Yu, F. R., Liu, J., Huang, T., Liu, Y. (2020). Deep reinforcement learning (DRL)-based device-to-device (D2D) caching with blockchain and mobile edge computing. *IEEE Transactions on Wireless Communications, 19(10),* 6469–6485. https://doi.org/10.1109/TWC.7693
29. Tang, C., Zhu, C., Wu, H., Li, Q., Rodrigues, J. J. P. C. (2021). Toward response time minimization considering energy consumption in caching-assisted vehicular edge computing. *IEEE Internet of Things Journal, 9(7),* 5051–5064.
30. Mu, S., Wang, F., Xiong, Z., Zhuang, X., Zhang, L. (2020). Optimal path strategy for the web computing under deep reinforcement learning. *International Journal of Web Information Systems, 16(5),* 529–544. https://doi.org/10.1108/IJWIS-08-2020-0055
31. Harper, F. M., Konstan, J. A. (2015). The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems, 5(4),* 1–19.

32. Li, Y., Zhou, A., Ma, X., Wang, S. (2021). Profit-aware edge server placement. *IEEE Internet of Things Journal, 9(1),* 55–67.

33. AlNagar, Y., Hosny, S., El-Sherif, A. A. (2019). Towards mobility-aware proactive caching for vehicular ad hoc networks. *2019 IEEE Wireless Communications and Networking Conference Workshop (WCNCW)*, pp. 1–6. Marrakech, Morocco.

34. Li, J., Zhao, J., Chen, P., Xia, Y., Li, F. et al. (2023). A multi-armed bandits learning-based approach to service caching in edge computing environment. *International Conference on Web Services*, pp. 3–17. Honolulu, USA.