Attention Mechanism-Aided Deep Reinforcement Learning for Dynamic Edge Caching

Ziyi Teng, *Student Member, IEEE,* Juan Fang, *Member, IEEE,* Huijing Yang, *Student Member, IEEE,* Lu Yu, Huijie Chen and Wei Xiang, *Senior Member, IEEE*

***Abstract*—The dynamic mechanism of joint proactive caching and cache replacement, which involves placing content items close to cache-enabled edge devices ahead of time until they are requested, is a promising technique for enhancing traffic offloading and relieving heavy network loads. However, due to limited edge cache capacity and wireless transmission resources, accurately predicting users' future requests and performing dynamic caching is crucial to effectively utilizing these limited resources. This paper investigates joint proactive caching and cache replacement strategies in a general mobile edge computing (MEC) network with multiple users under a cloud-edge-device collaboration architecture. The joint optimization problem is formulated as a markov decision process (MDP) problem with an infinite range of average network load costs, aiming to reduce network load traffic while efficiently utilizing the limited available transport resources. To address this issue, we design an Attention Weighted Deep Deterministic Policy Gradient (AWD2PG) model, which uses attention weights to allocate the number of channels from server to user, and applies deep deterministic policies on both user and server sides for Cache decision-making, so as to achieve the purpose of reducing network traffic load and improving network and cache resource utilization. We verify the convergence of the corresponding algorithms and demonstrate the effectiveness of the proposed AWD2PG strategy and benchmark in reducing network load and improving hit rate.**

***Index Terms*—Wireless network, edge caching, attention-weighted channel assignment, deep reinforcement learning.**

I. Introduction

With the rapid development of wireless access technology and the Internet of Things (IoT), a variety of smart mobile and IoT devices, such as smart phones and smart meter, are connected to wireless networks, which brings rapid growth service demand to mobile networks. Intelligent devices are highly dependent on computing, storage, and communication resources [1]. In traditional architectures, service requests require access to remote data centers (such as cloud servers) through core networks, which is insufficient to meet the requirements of delay sensitive and computationally intensive data services, posing significant challenges to the computing and caching capabilities of wireless communication systems [2].

Mobile edge computing (MEC) integrates the storage and processing capabilities of cloud computing into the edge of the network closing to user equipment (UE) and base station (BS), which is an effective way to reduce transmission distance and relieve core network load, and also provides an effective way to increase throughput and reduce latency simultaneously [3]. Despite these benefits, the limited storage capacity of edge nodes highlights the need to develop effective caching strategies to improve caching efficiency and minimize latency in edge networks.

Pure caching strategies have been extensively studied. According to the research method, it can be divided into two categories: *rule-based* and *model-based*. Traditional *rule-based* caching strategies such as least recently used (LRU), first in first out (FIFO) and their variants [3] are widely used due to their low computational complexity and ease of use. However, due to the lack of consideration of content popularity, these traditional caching strategies cannot adapt to real-time user requests when applied to edge nodes, thus exhibiting significant performance degradation. The *model-based* caching strategy [4]–[7] means that the content is dynamically updated according to the established target model. For example, Hachem *et al*. [4] and Shanmuga *et al*. [5] minimize caching cost and request time respectively by establishing models of content and wireless communication conditions. Furthermore, applied machine learning algorithms [6] [7] have demonstrated their effectiveness in edge cache optimization. Among them, Liu *et al*. [6] and Yan *et al*. [7] apply cache updates based on the popularity of the current request to reduce the traffic load on the backhaul link. However, the above cache update strategies are implemented "reactively", a *reactive cache* strategy that is utilized only when a request arrives. Therefore, unable to adapt to the variability of wireless channel conditions, resulting in inefficient bandwidth utilization.

Network traffic exhibits tidal effects in data transmission. For example, in the central business district, the network traffic on the back-haul link shows different trends during the day, and this tidal effect leads to lower bandwidth utilization when the network is idle [8] [9]. In order to fully utilize the bandwidth, the *proactive caching* strategy is introduced to deliver frequently accessed content before it is requested

Manuscript received 31 May, 2023; revised 14 Aug, 2023; accepted 13 October, 2023. This work is supported by National Natural Science Foundation of China (62202019,62276011,61202076), and supported by Beijing Municipal Natural Science Foundation (4192007), and Beijing University of Technology Project No. 2021C02, along with other government sponsors.

Ziyi Teng, Juan Fang, Huijing Yang and Huijie Chen are work with the Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China(e-mail: tengziyi@emails.bjut.edu.cn; fangjuan@bjut.edu.cn; yangkx@emails.bjut.edu.cn; lu.yu@my.jcu.edu.au, chenhuijie@bjut.edu.cn; *Corresponding author: Juan Fang*).

Lu Yu is with James Cook University, Cairns, QLD 4878, Australia(e-mail:lu.yu@my.jcu.edu.cn).

Wei Xiang is with La Trobe University, Melbourne, VIC 3086, Australia(e-mail: wei.xiang@latrobe.edu.au).

by users during idle network time, allowing flexible use of limited and time-varying communication resources.

Joint proactive caching and caching designs that can improve system performance by proactively transferring and replacing cached content during low-traffic times to meet future user needs have been extensively studied [10]–[13]. Somuyiwa *et al*. [10] proposes a threshold-based proactive caching strategy that requires causal knowledge of channel quality, content profile, and user access behavior. Meanwhile, Zheng *et al*. [11] decomposes the cache optimization problem into storage allocation and user allocation, and uses the Stackelberg game method to proactively cache content items to solve the problem of storage allocation during off-peak hours. Sun *et al*. [12] proposes a joint push and cache design to maximize bandwidth utilization. However, these proactive caching schemes above do not consider the limitation of the wireless transmission channel between the server and the user. On the other hand, Chen *et al*. [13] develops a multi-cast proactive content strategy based on structured deep learning, which considers the complex coupling of time-varying transmission capacity and proactive caching decisions between users, but only focuses on proactive caching at the user end, ignoring the its impact on the server side.

Based on the shortcomings of the above studies, we propose a joint proactive caching and cache replacement strategy that aims at proactively placing content items during network idle time, while taking into account the impact of the limited number of wireless channels and user caching decisions. However, we face the following technical challenges. First, each caching entity within the MEC system must determine which content items to proactively store in the cache and which ones to remove. Then, the user's proactively cached files must be forwarded through the server, and therefore, the user's proactive caching decisions must be taken into account at the server. Last but not least, due to the limited capacity of the transmission channel and the difference between the quality of the users' cache hit states and decision models, distributing the number of channels uniformly among the users may lead to a waste of network resources, which in turn affects the caching performance. Specifically, the factors affecting channel allocation can be classified into two categories: (1) user-related factors, such as user request frequency and user preference. User preferences play a significant role in determining the selection of proactively pushed files, as they are influenced by the file request probability. The request frequency, which represents the number of files requested per unit of time, further impacts the allocation and transmission of channels during the decision-making process. Ultimately, these factors have a direct impact on cache performance; (2) Model related factors such as model performance loss and model quality. The proactive selection of transferred files satisfying user requests relies on models that exhibit low loss and high quality. These models serve as crucial factors in determining the effectiveness of proactive file selection.

To address these issues described above, a dynamic caching strategy that combines proactive caching and cache replacement must continuously update the cached content as the environment changes, and reinforcement learning approaches can effectively solve complex online optimization problems and maximize the long-term reward without requiring prior knowledge of the network under consideration. Reinforcement learning approaches have been proven effective in cache optimization in previous work [14]–[16]. In this study, the attention-weighted depth deterministic policy gradient (AWD2PG) is proposed to solve the dynamic joint cache optimization problem between the user and server sides under limited transmission channels and time-varying and unobservable content popularity. Specifically, the contributions are summarized below.

- We formulate the joint proactive caching and cache replacement problems, and design dynamic caching mechanisms to minimize network load and improve the utilization of cache resources and network resources in the system.
- We transform the joint optimization problem into a Markov decision process (MDP) and apply deep deterministic strategies on both the user and server sides to handle large dimensions in state space and action space.
- We propose the attention weighted deep deterministic policy gradient (AWD2PG) framework, which uses an inter-layer deep deterministic strategy to address the inter-layer coupling caused by proactive caching. In particular, we use an attention mechanism to control the allocation of limited resource channels between users and server to accommodate the number of files transferred by user caching decisions for different time slots.

The remainder of this paper is organized as follows. Section II provides a summary of related work. Section III describes the system and the edge cache model. Section IV describes the problem formulation and analysis. The AWD2PG strategy is described in detail in Section V. The simulation results are presented in Section VI, followed by the conclusion in Section VII.

II. RELATED WORK

In this section, we mainly review the relevant research on proactive edge cache optimization using machine learning, especially the application of reinforcement learning in proactive proactive cache optimization.

A. Proactive Edge Caching

In practice, based on the popularity of the content or future user requests, the files are placed in advance to edge devices with caching enabled, which can significantly reduce the traffic load during peak times. Advances in machine learning and big data analytics have made it possible to accurately predict content popularity and user requests. Alqahtani *et al*. [16] proposed a proactive caching approach with offloading capabilities, which performs demand-aware offloading to meet the concurrent service dissemination requirements. Zhang *et al*. [17] designed a sequence-aware caching model to build a deep learning-based proactive caching policy to avoid the resource consumption of edge caching and achieve smaller traffic loads. Yu *et al*. [18] used the AutoEncoder (AE) model to predict request popularity and to perform proactive cache using the

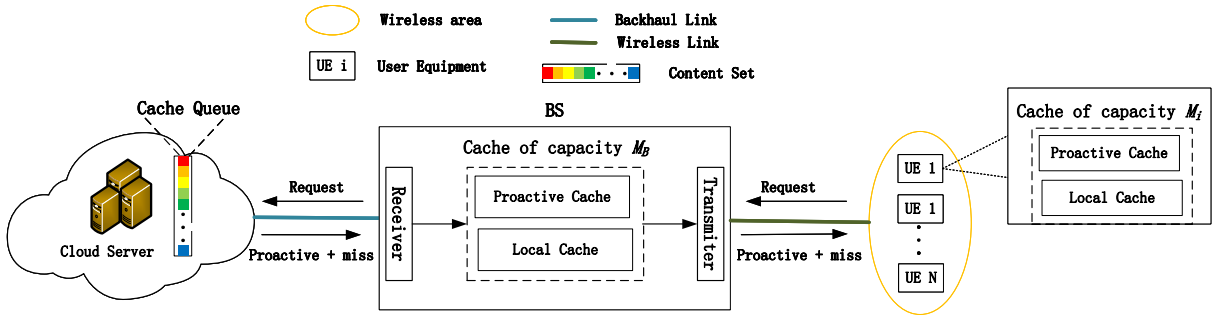

Fig. 1. The system architecture.

cooperative features of adjacent enabled cache devices. At the same time, there are various joint proactive caching strategies that aim to minimize traffic load [19], improve effective throughput [20], or reduce transmission energy consumption [21]. These strategies apply machine learning methods for proactive edge caching. However, they do not consider the network resources occupied by proactive caching and focus on the caching benefits of proactive caching.

While the effectiveness of proactive caching has received widespread attention, this study further explores proactive caching strategies between the server and user under wireless channel capacity constraints, which leads to network state complexity. Traditional caching approaches make it almost impossible to solve this complex problem. Therefore, we design a reinforcement learning-based algorithm to solve this problem.

B. Proactive Edge Caching with Applied Deep Reinforcement Learning

In practical applications, deploying deep reinforcement learning for online learning has been shown to be effective in managing complex and rapidly changing situations. Various studies have applied reinforcement learning techniques to address dynamic caching decisions [22]–[27]. Somuyiwa *et al*. [22] introduces an proactive caching strategy driven by reinforcement learning, aiming to reduce the system cost associated with wireless channel state and content download. Similarly, research in [23] optimizes system cost through reinforcement learning, while also considering how caching decisions in different time periods affect future content accessibility. Gao *et al*. [24] focuses on collaborative caching via reinforcement learning, with an emphasis on improving the efficiency of maximum distance separable (MDS) coding techniques. Sadeghi *et al*. [25], an important insight emerges: intra-slot caching decisions affect immediate cost and future cache availability. This has led to the application of reinforcement learning to solve caching problems involving constantly changing and stochastic costs. Also, Hebatullah *et al*. [26] introduces a model-independent meta-reinforcement learning algorithm to accommodate dynamic cache management and accommodate fluctuations in content popularity caused by changes in associated devices. Likewise, Zhou *et al*. [27] utilizes multi-agent reinforcement learning to address the challenges of cooperative and proactive caching, emphasizing the minimization of average download latency.

Different from and complementary to the existing work which focus primarily on proactive caching at the local or server side, we consider dealing with the coupling problem of the proactive caching scheme between the server and user side under the Limited number of wireless channels.

III. SYSTEM MODEL

A. System Architecture

The general model of the MEC network for edge caching considered in this study, including cloud, BS and N user equipment (UE), is shown in Fig.1. All users represented as $\mathcal{N} = \{1, 2, \ldots, N\}$. Both users and BS equipped with servers have certain computing and caching capabilities. The BS includes a receiver and a transmitter for receiving the content transmitted by the cloud center and transmitting the content to the user. That is, the cloud server transmits files to users through BS. Let $\mathcal{F} = \{1, 2, 3, \ldots, F\}$ denote the content set of all these accessible from the cloud server. The system operates in infinite time slots, divided into L time slots, denoted as $T = 1, 2, 3, \ldots,$. In each time slot t, each UE-i generates a different number of requests, and it is assumed that users do not make repeated requests for the same file.

At each time slot t, for each user UE-i, if the request is saved in the local cache, it will immediately return; Otherwise, users must access the missed content from the server. Similarly, when the MEC server cannot meet the requirements, it is accessed through a backhaul link from the cloud data center. The cache capacity of UE-i is M_i, MEC server is M_B. In general, $M_i \leq M_B \leq F$. In each time slot t, the wireless transmission channel between the MEC server and users is limited, and the total number of available channels is represented as $C(t)$, which means that all users within the base station range can transmit up to $C(t)$ files.

B. Content Popularity and User Request Model

1) *Content popularity*: In our model, we represent the user's preference, which indicates the popularity of the

requested content, using the Mandelbrot-Zipf (M-Zipf) distribution [28].

$$P_f = \frac{(o_f + \tau)^{-\beta}}{\sum_{i \in \mathcal{F}(o_i+\tau)^{-\beta}}}, \forall f \in \mathcal{F}, \tag{1}$$

where O_f is the descending content popularity ranking of content f, and τ and β respectively denote the plateau factor and deviation factor, where $\tau \geq 0$. Furthermore, it is assumed that the popularity of content changes slowly over a relatively long period.

2) *User request model*: During the time slot t, each UE-i generates content requests with an arrival rate λ_i, and the number of user requests follows the *Poisson* distribution [1] [7] [15] [29]:

$$P(X(t) = k) = \frac{\lambda_i^k}{k!} e^{-\lambda_i}, \tag{2}$$

where λ_i indicates the frequency of requests from UE-i and k indicates the number of requests made by the user in time slot t.

C. Model Validation and Insights

The goal of this section is to verify the effectiveness of using Zipf distribution to simulate user request probability and Poisson distribution to simulate cache request arrival rate when the number of users is large.

Based on equation (1), in order to verify the validity, we can approximate the probability as:

$$P_f \approx \frac{(o_f + \tau)^{-\beta}}{\int_{i=1}^{\infty} (o_i + \tau)^{-\beta} di}$$

By employing integral approximation and assuming a large number of users, we can derive the following approximation:

$$P_f \approx C \cdot (o_f + \tau)^{-\beta}$$

Here, C represents a constant. This approximation serves as evidence that the Zipf distribution can effectively capture the popularity of user requests when the number of users is large.

In order to prove the effectiveness of *Poisson* distribution when the number of user requests increases, we need to consider the limit situation when the number of user requests increases. In this case, the arrival rate parameter λ_i also increases significantly. It is known that when the event rate increases, the Poisson distribution gradually approaches the normal distribution. Therefore, when the number of user requests is large, Poisson distribution can effectively represent the arrival rate of user requests.

D. Dynamic Cache Handling Process

Due to the limited cache resources of MEC servers and user devices, a joint proactive cache and cache replacement strategy is proposed to reduce network costs caused by missing requests. Fig.2 depicts the decision-making process within a time slot t. Assuming the duration of a time slot $t$is long enough to make dynamic caching decisions. The edge node decision-making process with caching enabled can be divided into three stages within a time slot. The first phase is the "`user request`" phase, in which the user creates content requests, checks the local cache for cache misses, and then sends missing requests to the server. The second phase is the "`dynamic caching decision`" phase. According to the current request and miss file state, the proactive cache files be selected from the content library. If the number of proactive transmission files is greater than the number of free caches on a wireless transmission channel constraints, the replacement policy decides which contents must be replaced from the cache. Finally, in the "`cache update`" phase, in which the cache is dynamically updated prepare for the next phase of user requests based on the cache policy obtained in the previous phase, and the policy performance of the current time slot is evaluated. The performance is determined by the network traffic load, which primarily includes the load of request misses in the first phase and the network load caused by the proactive caching policy in the second phase.

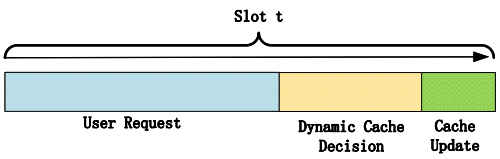

Fig. 2. The decision-making process in time slot t.

E. System Model

The user request arrival rate follows the Poisson process (2), with each UE-i submitting at least one request in time slot t. The set of requested files is $q_i(t) = \{q_i^f(t) \in \{0,1\} | f \in \mathcal{F}\}$, where $q_i^f(t) = 1$ means that the user requested f and $q_i^f(t) = 0$ means that the file f was not be requested in time slot t. Lets $CU_i(t) = \{CU_i^f \in \{0,1\} | f \in \mathcal{F}\}$ denotes the cache state of UE-i in time slot t, where $CU_i^f(t) = 1$ means that UE-i has stored content f; otherwise, $CU_i^f(t) = 0$ means that f has not been placed in the cache. To meet the cache space size limit, the following must be satisfied.

$$\sum_{f \in \mathcal{F}} CU_i^f \leq M_i \tag{3}$$

If the requested file $q_i^f(t)$ is saved in the cache, i.e., $q_i^f(t) \in CU_i(t)$, the local cache hits immediately; otherwise, the user accesses the missing content from the MEC server. Similarly, for the server side, let $CU_B(t) = \{CU_B^f(t) \in \{0,1\} | f \in \mathcal{F}\}$ indicate the cache state at the server in time slot t, where $CU_B^f(t) = 1$ means that the server stores content f and $CU_B^f(t) = 0$ means that f has not been placed in the cache. Thus, under the limit of the server-side cache size, the following holds.

$$\sum_{f \in \mathcal{F}} CU_B^f \leq M_B \tag{4}$$

The MEC server receives missing requests from all users in the coverage area at each time slot, denoted as $Q^G(t)$. The MEC server needs to send missing requests and proactively

cached files to users. Therefore, the wireless channel resource occupation between the MEC server and the user includes two aspects: one is the file transfer caused by proactive caching, and the other is the file transfer caused by file missing. Its file sets can be expressed as $\mathbb{PT}_i = \{PT_i^f(t) \in \{0,1\} | f \in \mathcal{F}\}$ and $\mathbb{RT}_i = \{RT_i^f(t) \in \{0,1\} | f \in \mathcal{F}\}$, respectively, where $PT_i^f = 1$ means that the content f need proactively transmitted to UE-i; otherwise if $PT_i^f = 0$, it means no need to transfer; $RT_i^f = 1$ indicates reactive transmission caused by missing requests; otherwise if $RT_i^f = 0$, it means no need to transfer. In order to meet the user's request at the end of each time slot, the reactive transmission from the MEC server to the user should meet the following conditions.

$$RT_i^f(t) = q_i^f(t)(1 - CU_i^f(t)) \tag{5}$$

The proactive and reactive transfer file sets on the server side are represented as $\mathbb{PT}_B(t) = \{PT_B^f(t) \in \{0,1\} | f \in \mathcal{F}\}$ and $\mathbb{RT}_B = \{RT_B^f(t) \in \{0,1\} | f \in \mathcal{F}\}$, respectively. The reactive transmission state of file f from the cloud server to BS caused by a missing file is represented as follows.

$$RT_B^f(t) = (Q^G(t) = f)(1 - CU_B^f(t)) \tag{6}$$

If the content of the proactive cache decision transmission at UE-i is already cached in $CU_i(t)$, it does not need to be transmitted through the server; therefore, we have

$$PT_i^f(t) = (1 - q_i^f(t))(1 - CU_i^f(t)). \tag{7}$$

Similarly, proactive cache file transfer to the server side is denoted as follows.

$$PT_B^f(t) = \Big(1 - (Q^G(t) = f)\Big)(1 - CU_B^f(t)) \tag{8}$$

Considering that a file f can only be transmitted at most once in a time slot t, we have

$$RT_i^f(t) PT_i^f(t) = 0, \tag{9}$$

$$RT_B^f(t) PT_B^f(t) = 0. \tag{10}$$

Taking into account the limitation of the number of wireless transmission channels between BS and users. Therefore, it is necessary to satisfy

$$\sum_{i \in \mathcal{N}} \sum_{f \in \mathcal{F}} \Big(PT_i^f(t) + RT_i^f(t)\Big) \leq C(t), \tag{11}$$

where $C(t)$ denotes the maximum number of wireless channel transmissions allowed during time slot t.

Dynamic cache policies are used to improve the utilization of cache resources. Let $z_i^+(t) = [a_{RT_i}^+(t)]_{RT_i=1}^{|RT_i(t)|}$ decide which files in $RT_i(t)$ should be saved in the UE-i cache at time t, where $a_{RT_i}^+(t) = 1$ indicates that the file $f_{RT_i} \in RT_i(t)$ should be saved to $CU_i(t)$; otherwise, if $a_{RT_i}^+(t) = 0$ means that the file should be discarded entirely. Thus, the new file saved to the user-side cache at time slot t is represented as $a_i^+(t) = z_i^+(t) \cup PT_i(t)$. In addition, when the number of new files is greater than the free space in the cache, the cached files must be eviction. Let $a_i^-(t) = [a_{cu_i}^-(t)]_{cu_i=1}^{M_i}$ denote which files from the cache $CU_i(t)$ should be selected for eviction from UE-i at time slot t, where $a_{cu_i}^-(t) = 1$ means that the file $f_{cu_i} \in CU_i(t)$ should be evicted from $CU_i(t)$; otherwise, if $a_{cu_i}^-(t) = 0$, the file should be retained. We can also obtain the following cache capability constraint of UE-i

$$\sum_{cu_i=1}^{M_i} a_{cu_i}^-(t) = \sum_{RT_i=1}^{|RT_i(t)|} a_{RT_i}^+(t) + |PT_i(t)|, \tag{12}$$

which indicates that the number of files deleted from the cache is consistent with the number of files placed in the cache.

The decisions for $a_i^-(t)$ and $a_i^+(t)$ are made in the final phase "`dynamic caching decision`" of the dynamic cache mechanism processing. This dynamic caching mechanism is also present on the MEC server.

Let $a_B^-(t) = [a_{CB}^-(t)]_{CB=1}^{M_B}$ decide which files $f_{CB} \in CB(t)$ should be evicted from the server-side cache $CB(t)$ at time slot t; where $a_{CB}^-(t) = 1$ indicates that the file $f_{CB} \in CB(t)$ should be evicted from the server-side cache $CB(t)$ at time slot t; otherwise, if $a_{CB}^-(t) = 0$ indicates that it still needs to be preserved. Let $z_B^+(t) = [a_{RT_B}^+(t)]_{RT_B=1}^{|RT_B(t)|}$ denotes which files in $RT_B(t)$ should be placed in the MEC server cache at time t, where $a_{RT_B}^+(t) = 1$ means that the file $f_{RT_B} \in RT_B(t)$ should be stored; otherwise, if $a_{RT_B}^+(t) = 0$, it should be discarded. Therefore, the set of files that should be replaced in the cache of the MEC server during the time period t is expressed as $a_B^+(t) = z_B^+(t) \cup PT_B(t)$. Due to the limitation of cache capacity, it should satisfy

$$\sum_{CB=1}^{M_B} a_{CB}^-(t) = \sum_{rt_B=1}^{|RT_B(t)|} a_{rt_B}^+ + |PT_B(t)|. \tag{13}$$

In other words, the number of files deleted from the MEC server should match the number of files added to the cache. On the user side, the dynamic caching action of joint proactive caching and cache replacement is represented as follows.

$$a_i(t) = \{a_i^+(t), a_i^-(t)\} \tag{14}$$

Similarly, on the MEC server, dynamic caching action is represented as follows.

$$a_B(t) = \{a_B^+(t), a_B^-(t)\} \tag{15}$$

After dynamically caching action a_i, the user's cache state $CU_i(t)$ is updated to $CU_i(t+1)$ at time slot t in preparation for the user request in the next time slot. This cache update process also occurs in the second phase "cache update" of the dynamic cache mechanism processing. Similarly, the cache state on the server side changes from $CU_B(t)$ to $CU_B(t+1)$ under the action $a_B(t)$.

For ease of understanding, the notation used in this article is summarized in Table I.

F. Network Traffic Load

The network load $Ta(t)$ of each time slot t is defined as the number of files transmitted in the network, including the two parts of proactive push file transmission and transmission caused by cache miss, which are recorded as $Ta^1(t)$ and $Ta^2(t)$. According to the dynamic cache

TABLE I
MODELING PARAMETERS AND NOTATIONS

Symbol	Description
t	Index of the time slot.
$\mathcal{N}=\{1,2,\ldots,N\}$	Set of user labels.
$\mathcal{F}=\{1,2,\ldots,F\}$	Set of all contents.
M_i, M_B	Cache capacities of UE-i and the MEC server, respectively.
$q_i(t), q_i^f$	Request set of UE-i at time slot t.
$Q^G(t)$	Missing requests from all users at time slot t.
$C(t)$	Number of available channels in time slot t between the MEC server and the user.
$CU_i(t)$ /$CB(t)$	Cache states of UE-i and the MEC server at time slot t, respectively.
$PT_i^f(t)/RT_i^f(t)$, $PT_B^f(t)/RT_B^f(t)$	Proactive/reactive file of UE-i and BS for file f at time slot t, respectively.
$\mathbb{PT}_i$, $\mathbb{RT}_i$	Proctive transfers and missing transfers file sets for $UE-i$, respectively.
$a_i=\{a_i^+(t),a_i^-(t)\}$, $a_B=\{a_B^+(t),a_B^-(t)\}$	Dynamic caching actions of UE-i and the MEC server at time slot t, respectively.
$a_{rt_i}^+(t)$, $a_{cu_i}^-(t)$	Indicators for whether the UE-i adds or deletes the file from $RT_i(t)/CU_i(t)$, respectively.
$a_{rt_B}^+(t)$, $a_{cu_B}^-(t)$	Indicators for whether the MEC server adds or deletes the file from $RT_B(t)/CU_B(t)$, respectively.
$z_i^+(t)=[a_{RT_i}^+(t)]_{RT_i=1}^{\lvert RT_i(t)\rvert}$,$z_B^+(t)=[a_{RT_B}^+(t)]_{RT_B=1}^{\lvert RT_B(t)\rvert}$	Indicates which files from $RT_i(t)/RT_B(t)$ and should be saved to UE-i and BS cache at t, respectively.
$Ta^1(t)$, $Ta^2(t)$	Indicates the traffic load caused by missing requests and the traffic load caused by proactive transmission, respectively.
$Ta_i(t), Ta_B(t)$	Indicates the traffic load caused by UE-i and MEC server request loss and proactive transmission, respectively.
ω_i^p/ω_i^m , ω_B^p/ω_B^m	Indicates the proportions of active transmission and request loss traffic loads on the user side and the MEC server side, respectively.

handing processing, the traffic load caused by missing requests occurs in the "`user request`" phase, and the traffic load caused by proactive transmission occurs in the "`Dynamic Cache Decision`" phase. The missing request in the time period $t-1$ means that its cached content $CU_i(t-1)$ or $CU_B(t-1)$ cannot meet the request, which needs to be forwarded to the upper level for processing. Therefore, the file loss traffic loads on the user side and the MEC server side are expressed respectively as

$$Ta_i^1(t)=2f_0\sum_{f\in\mathcal{F}}RT_i^f(t), \tag{16}$$

$$Ta_B^1(t)=2f_0\sum_{f\in\mathcal{F}}RT_B^f(t), \tag{17}$$

where f_0 represents the size of the requested contents. The traffic load caused by proactively caching at the user and MEC server side are respectively expressed as follows.

$$Ta_i^2(t)=f_0\sum_{f\in\mathcal{F}}PT_i^f(t), \tag{18}$$

$$Ta_B^2(t)=f_0\sum_{f\in\mathcal{F}}PT_B^f(t). \tag{19}$$

Therefore, the network load caused by UE-i is denoted as

$$Ta_i(t)=\omega_i^m Ta_i^1(t)+\omega_i^p Ta_i^2(t), \tag{20}$$

where ω_i^m and ω_i^p respectively denote the user-side traffic load shares, where $\omega_i^m+\omega_i^p=1$. Similarly, the traffic load between the cloud server and the MEC server is expressed as

$$Ta_B(t)=\omega_B^m Ta_B^1(t)+\omega_B^p Ta_B^2(t), \tag{21}$$

where ω_B^m and ω_B^P respectively denote the weights of the server-side traffic load of request misses and the proactive cache traffic load, where $\omega_B^m+\omega_B^P=1$.

To achieve the objective of minimizing the network load $Ta(t)$ within the system, which is represented by the expression $Ta(t)=\sum_{i=1}^N Ta_i(t)+Ta_B(t)$, the method of problem decomposition is utilized. Specifically, this approach involves solving problems (20) and (21) separately.

IV. Problem Formation and Analysis

A. Problem Formation

To effectively utilize the limited cache resources in MEC systems, the joint proactive caching and cache replacement problem can be formulated as an optimization problem. That is, at each time slot t, each edge cache node must decide which files to cache and replace. The caching decisions at the user side and the MEC server side are defined as μ_i and μ_B, respectively. In order to reduce the network load, a traffic load cost is defined to measure the performance of the dynamic caching strategy for better utilization of network resources. Considering the uncertainty of the UE request, the average transmission traffic load of the user side in the time slot t is $E[\omega_i^m Ta_i^1(t)+\omega_i^p Ta_i^2(t)]$, where E is the expected requirement of UE-i. In order to obtain the optimal transmission cost in an infinite time range, the average network transmission load cost of UE-i is defined as follows

$$\psi(\mu_i)=\lim_{T\to+\infty}\frac{1}{T}\sum_{t=1}^{T}E[\omega_i^m Ta_i^1(t)+\omega_i^p Ta_i^2(t)]. \tag{22}$$

Correspondingly, the average network transmission load cost between the cloud server and the MEC server is defined as follows.

$$\psi(\mu_B)=\lim_{T\to+\infty}\frac{1}{T}\sum_{t=1}^{T}E[\omega_B^m Ta_B^1(t)+\omega_B^p Ta_B^2(t)] \tag{23}$$

To minimize the average network transmission load for UE-i, the optimization problem is formulated as

$$\begin{aligned}&\rho 1:\psi_i^*=\min_{\mu_i}\psi(\mu_i)\\&s.t.(3)(5)(7)(9)(11)(12),\end{aligned} \tag{24}$$

where ψ_i^* is denoted as the optimal time-averaged network load cost from the MEC server side to UE-i; constraint (3) is the cache size constraints for the user; constraints (5), (7), and (9) denote the limits on proactive and reactive file transfers; constraints (11) denotes the wireless channel capacity limit that should be satisfied by the transferred files, and (12)

denotes the size constraints of dynamic cache decision. The average traffic load problem from the cloud service center to the server side can be formulated as

$$\begin{aligned} &\rho 2: \psi_B^* = \min_{\mu_B} \psi(\mu_B) \\ &s.t. (4)(6)(8)(10)(13), \end{aligned} \tag{25}$$

where ψ_B^* is denoted as the optimal time-averaged network load cost from the cloud server to the MEC server side; constraint (4) is the cache size constraints for the MEC server; constraints (6), (8), and (10) denote the limits on proactive and reactive file transfer on the MEC server; and (13) denotes the capacity constraints of dynamic cache decision on the MEC server side.

Note that optimization problems (24) and (25) are characterized by the following facts and technical challenges.

- The varying popularity and arrival rates of user requests create a high-dimensional dynamic network, and the caching policy of each caching node must solve quickly.
- The solutions to problems (5), (7) and (6), (8) are all dynamic policies over time, rather than a transient ones. In addition, the number of transmittable channels and the number of local missing requests are variable within each time slot t, so the dimensions of $PT_i^f(t)$ and $RT_i^f(t)$, and those of $PT_B^f(t)$ and $RT_B^f(t)$, are also time-varying.
- Due to the heterogeneity of users' decision model quality and the number of requests, it is unreasonable to share the limited number of channels equally among users.
- The file input on the MEC server side is determined by user caching decisions, and user caching decisions also affect the input state on the server side. Moreover, the state of the MEC server and user conform to contextual chain property over time.

B. Problem Recasting

The cache optimization problem is difficult to solve directly due to the interaction between the user-side and server-side caching policies, as well as the complexity of individual caching policies without knowledge of user request popularity and request transfer probability. To overcome the above technical challenges, the underlying optimization problem is converted into an MDP problem, to find the optimal policy for minimizing the network load. The MDP consists of four components, namely, the state space, action space, state transfer probability, and reward. Specifically, the descriptions of problems (24) and (25) are denoted as $< S_i, A_i, P_i, R_i >$ and $< S_B, A_B, P_B, R_B >$, respectively.

- **State**: The set of states respectively describes the request and cache states on the user side and the MEC server side. Considering the time variable t, the user state and server-side state can be respectively represented as $S_i = \{s_i(t)|t = 0, 1, 2, \dots\}$ and $S_B = \{s_B(t)|t = 0, 1, 2, \dots\}$. Based on the necessary information required for the dynamic caching of action, $s_i(t) = \{CU_i(t), q_i(t)\}$ and $s_B(t) = \{CU_B(t), Q^G(t)\}$ are defined, where $Q^G(t)$ is represented as the missing files of all users.
- **Action**: The cache actions of the user and server side are to update their cache space according to the cache policy. According to the problem statement provided previously, a_i and a_B already exist, as shown in (14) and (15). Therefore, the user-side and server-side cache action can be respectively represented as $A_i = \{a_i(t)|t = 0, 1, 2, \dots\}$ and $A_B = \{a_B(t)|t = 0, 1, 2, \dots\}$. After the user-side and server-side caches execute an action, the cache state is updated as $CU_i(t+1) = CU_i(t) \backslash a_i^-(t) \cup a_i^+(t)$, $CU_B(t+1) = CU_B(t) \backslash a_B^-(t) \cup a_B^+(t)$.
- **Transfer probability** P: The state transfer probability describes the transfer of the system to the next state after performing the action $a_i(t)$ or $a_B(t)$ in the current state. For problems (24) and (25), the state transfer probability can be expressed as $P_i\Big(s_i(t), s_i'(t)|a_i(t)\Big) \in \mathcal{P}^i, \forall s_i(t), s_i'(t) \in S_i$ and $P_B\Big(s_B(t), s_B'(t)|a_B(t)\Big) \in \mathcal{P}^B, \forall s_B(t), s_B'(t) \in S_B$, where $s_i'(t)$ and $s_B'(t)$ denote the cache state after taking action in states $s_i(t)$ and $s_B(t)$, respectively.
- **Reward**: The reward is a function of the state and action, and it measures the effect of the agent's action in a given state. Because the goal of the optimization is to reduce the network traffic load, the reward is defined as the cost of the file transfer, as follows.

$$r_i(s_i(t), a_i(t)) = -Ta_i(s_i(t), a_i(t)) \tag{26}$$

$$r_B(s_B(t), a_B(t)) = -Ta_B(s_B(t), a_B(t)) \tag{27}$$

The cumulative rewards from time t can respectively be expressed as follows.

$$\overline{r}_i \triangleq \sum_{t=0}^{T} r_i(s_i(t), a_i(t)) \tag{28}$$

$$\overline{r}_B \triangleq \sum_{t=0}^{T} r_B(s_B(t), a_B(t)) \tag{29}$$

C. Reinforcement Learning Formation

According to equations (28) and (29), the agent's goal is to learn a caching policy that maximizes the cumulative reward value. The user-side and MEC server-side policy functions are respectively defined as $\mu_i \colon s_i \to a_i$ and $\mu_B \colon s_B \to a_B$, which implements a mapping from the current system state to a series of actions. The action $a_i(t) = \mu_i[s_i(t)]$ determines what should be stored in the cache in the current state $s_i(t)$ under the policies $\mu_i(*)$. The goal of user side cache policy is to find the best policy to maximize the cumulative reward (state value function), which is expressed as follows.

$$V_{\mu_i}(s_i(t)) = \lim_{T\to\infty} \mathbb{E}[\sum_{\tau=t}^{T} \gamma^{t-1} r_i(s_i(t), a_i(t)|s_i(0) = s_i)] \tag{30}$$

This represents the average network traffic load cost incurred over an infinite time, where $\mathbb{E}$ denotes the expectation and the discount factor is $\gamma \in [0, 1)$. Action $a_i(t)$ affects the user-side and server-side cache states in future time slots, so past and present actions always have an impact on future costs.

Due to the nature of Markov models, by using the Bellman equation with recurrence relations [30], the value function (30) can also be expressed as follows.

$$V_{\mu_i}(s_i(t)) = \sum_{a_i \in A_i} \mu_i\Big(a_i(t)|s_i(t)\Big)\Big\{r_i(s_i(t), a_i(t)) + \gamma \sum_{s_i'} P_i\Big(s_i'(t)|s_i(t), a_i(t)\Big)V(s_i'(t))\Big\} \tag{31}$$

Thus, the optimal strategies ψ_i^* and ψ_B^* for problems (24) and (25) can be derived as objective strategy functions based on the Bellman equation, respectively.

$$\psi_i^* = \arg\max_{a_i \in A_i} \sum_{s_i(t+1) \in S_i} P_i(s_i(t), s_i'(t)|a_i(t)) \cdot \Big(r_i(s_i(t), a_i(t)) + \gamma V_{\mu_i}(s_i(t+1))\Big) \tag{32}$$

$$\psi_B^* = \arg\max_{a_B \in A_B} \sum_{s_B(t+1) \in S_B} P_B(s_B(t), s_B'(t)|a_B(t)) \cdot \Big(r_B(s_B(t), a_B(t)) + \gamma V_{\mu_B}(s_B(t+1))\Big) \tag{33}$$

The optimal caching policy ψ_i^* on the user side yields the optimized action $a_i(t)$ at state $s_i(t)$. The number of transmitted files caused by an action $a_i(t)$ at time slot t varies with $C(t)$. As a result, traditional optimization techniques such as dynamic programming are ineffective for solving the problem considered in this study.

V. Overview Design Of AWD2PG

To address the aforementioned optimal issue (32)(33), the AWD2PG is proposed, which is combined with the dynamic cache decision process as Section III-C and consists of four major phases: *the user request* phase, *the attention channel allocation* phase, *the local cache decision* phase and *the server cache decision* phase. In particular, Section V-A presents the overall process of the proposed AWD2PG at time t. Section V-B and Section V-C describe the channel allocation method for attention weights, as well as the user-side caching policy. Finally, Section V-D describes the server-side caching strategy.

A. Whole Process

The overview process of the proposed AWD2PG at time t is presented in Fig. 3.

1) *User Request Phase*: In this phase, the user sends the request and checks the local cache, then sends the missing files and user model related factors to the server.
2) *Attention Channel Allocation Phase*: In this phase, the MEC server-side collects relevant factors for each UE (e.g., the number of user misses, the average loss, and the average reward value). Then, using an attention mechanism, different channel allocation weights are given to all devices within the coverage of the base station. Finally, the obtained channel numbers are distributed to corresponding devices for user-side caching decisions.
3) *Local Cache Decision Phase*: Due to the continuous processing operation space of the cache decision, at this stage, the user side adopts the deterministic policy gradient (DDPG) model with high sampling efficiency, generates a local cache policy according to the number of allocated channels, and dynamically updates the cache.
4) *MEC server Cache Decision Phase*: All users in the coverage area send proactive cache decision actions and missing files to the server, which incorporates the current cache state on the server side and uses the twin delayed DDPG (TD3) [31] model to make cache decisions to update the cache.

B. Weighted Channel Allocation

Under unicast between the MEC server and the user, all users share the available channels equally. However, considering the difference in the performance of the UE's local decision-making model, it is unreasonable to allocate all channels equally. Therefore, a weighted form (ie., weighted channel allocation) is considered. In this model, a reward $\overline{r}_i((s_i(t), a_i(t))$ and some device-related metrics are used to evaluate the number of channels allocated to user devices. This happens in *weighted channel allocation* stage (shown in figure 3).

Problem Formulation: The corresponding weighted channel assignment problem can be expressed as follows,

$$c_i(t) = w_i C(t), \tag{34}$$

where w_i is the weighting factor for UE-i, which is used to calculate the proportion of channels allocated to users. The weights w_i for UE-i are calculated using the average reward, the average loss of the critic/actor network, the number of missing requests, and the batch size.

- **Average reward**: The average reward $\overline{r}i(t)$ of UE-i is the average of all cache decision rewards in time slot $t-1$.
- **The average loss of critic/actor network**: The average loss value of the critic and the actor network for UE-i respectively express as $\overline{L}_c^i(t)$ and $\overline{L}_a^i(t)$, which is the average of the local training model losses over the $t-1$ time slot.
- **The number of missing requests**: The number of missing requests for UE-i in time slot t is m_i^t.
- **Batch size**: The batch size of UE-i is b_i; The larger the batch size, the more data will be involved in the training process of the local model.

In the proposed model, the Scaled dot-product attention is utilized, and the evaluation metrics are defined as "Keys" and "Values" in the attention mechanism. Specifically, "Keys" are expressed as $K_i = [\overline{r}_i(t), \overline{L}_C^i(t), \overline{L}_A^i(t), m_i^t, b_i]$. The objective is to develop a more capable agent that can maximize rewards while minimizing losses. Additionally, users believe that a higher number of lost requests necessitates a higher number of channels. As a result, the **Query** is designed as follows.

$$Q = [\max_i(\overline{r}_i(t)), \min_i(\overline{L}_c^i(t)), \min_i(\overline{L}_A^i(t)), \max_i(m_i^t), \max_i(b_i)] \tag{35}$$

In the attention weight distribution stage, the input on the MEC server side includes query $\mathbf{Q}$, keys K_i. Calculate the dot

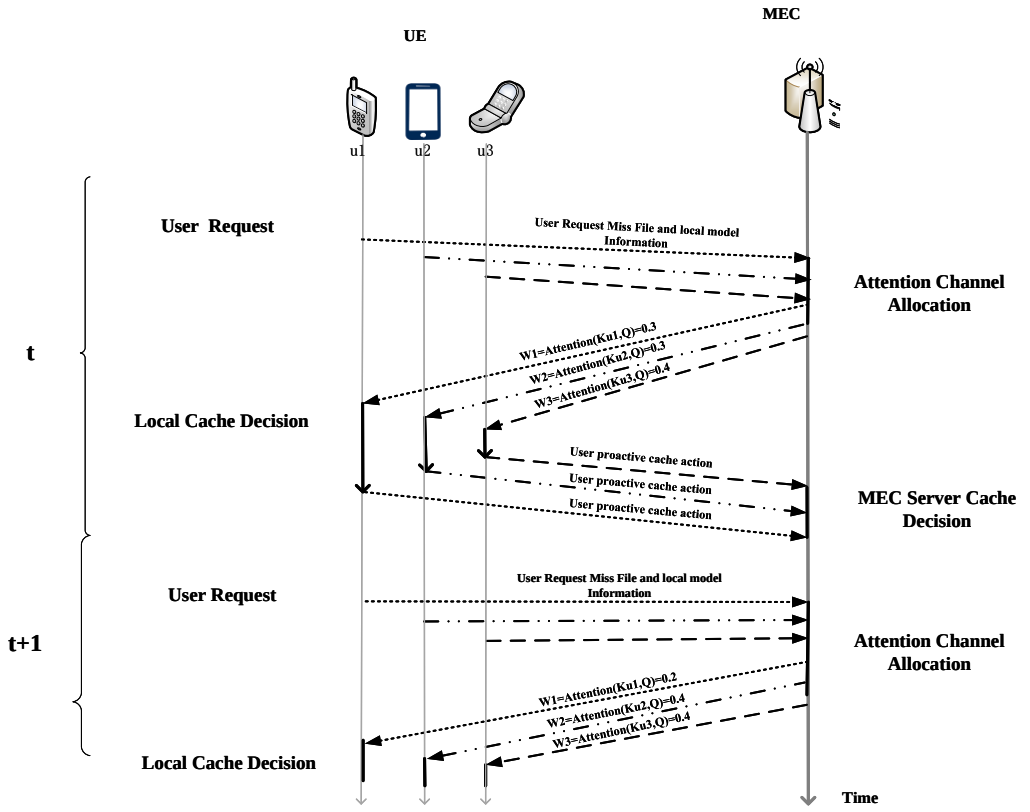

Fig. 3. The whole process of the proposed AWD2PG framework at time slot t.

product of query and all keys, divide by $\sqrt{d_k}$, and then use the ***softmax*** function to get the weight of the value, where $\sqrt{d_k}$ is the dimension of keys K_i. Therefore, for weight factors w_i, we have

$$w_i = Attention(Q, K_i) = softmax(\frac{QK_i^T}{\sqrt{d_k}}), \forall i \in \mathcal{N}. \quad (36)$$

C. Local Cache Decision

The user receives the number of channels allocated (34) in the *Local Caching Decision* phase. The user then makes a caching decision based on this, i.e., decides to proactively transmit and cache the content to be replaced. However, due to the large state space and action space, it is difficult to obtain an optimal policy for problem (32).

In order to tackle the developed MDP problem, we have employed a reinforcement learning algorithm to execute dynamic caching decisions based on the designated state and action spaces. Firstly, within our scenario, the deployed agents on edge devices are confronted with the task of making decisions regarding caching or transmitting files. This decision-making process involves selecting from a wide range of options, thereby transforming it into a continuous action problem. Secondly, the agents deployed on the edge devices must engage with the environment for training purposes. Given the limited computational or storage resources available on these edge devices, it becomes imperative to deploy a model that is both straightforward and easy to implement. In related studies, we discovered that the DDPG algorithm was effectively utilized for distributed reinforcement learning training in [6]. Likewise, in [9], the Deep Q-Network (DQN) algorithm was employed to address the decomposed user caching optimization subproblem. Drawing upon these findings, we have decided to utilize the DDPG algorithm for executing dynamic caching decisions within our scenario.

DDPG adopts the actor-critic method, the critic network $Q_i(s_i, a_i|\theta^{Q_i})$ evaluates the value of Q, and the actor network evaluates the policy function $\mu_i(s_i|\theta^{\mu_i})$. The training parameters of critic network and action network are θ^{Q_i} and θ^{μ_i} respectively.

In each time slot, the user sends out requests and gets the missing information of the current request according to the current cache state. After receiving the number of channels allocated by the MEC server side, combine the user request of the current slot with the cached state $CU_i(t)$ to generate the current user state $s_i(t)$ and feed it to the actor network. The actor network equals a parameterized actor function $a_i(t) = \mu_i(s_i(t)|\theta^{\mu_i})$, which determinedly mapped the state to a specific action. For the intelligence to fully explore the environment, exploration and exploitation are balanced in this work by adding a Gaussian noise vector to the strategy output [30], i.e.,

$$a_i(t) = \mu_i(s_i(t)|\theta^{\mu_i}) + n_0(t)|_{n_0(t)\sim\mathcal{N}(0,\sigma^2)}, \quad (37)$$

where $n_0(t)$ is a Gaussian-distributed noise vector with a mean of 0 and a variance of 2. Under action $a_i(t)$, the user's cache

state $CU_i(t)$ in time slot t is updated to $CU_i(t+1)$. The obtained action $a_i(t)$ is then send to the critic network to obtain the corresponding Q-value, denoted as $Q_i(s_i(t), a_i(t)|\theta^{Q_i})$, and the Q-value can be further expressed as follows.

$$Q_i(s_i(t), a_i(t)|\theta^{Q_i}) = \mathbb{E}[\sum_{\tau=0}^{+\infty} \gamma^{t-1} r_i(s_i(t), a_i(t))] \tag{38}$$

Experience playback is used to improve the training stability on the user side, and the dataset in its experience pool can be represented as follows.

$$\Omega = (s_i(t), a_i(t), r_i(t), s_i'(t)) \tag{39}$$

Furthermore, the networks $\mu_i(\cdot|\theta^{\mu_i})$ and $Q_i(\cdot|\theta^{Q_i})$ are referred to as online networks, and there are two identical counterparts for greater stability and faster convergence. The corresponding networks are the target actor network $\mu_i(s_i|\theta^{\mu_i'})$ and the target critic network $Q_i(s_i, a_i|\theta^{Q_i'})$, where $\theta^{\mu_i'}$ and $\theta^{Q_i'}$ are the corresponding neural network parameter. The target network is a time-delayed copy of the original network that can track the learned network and softly update the target network model parameters. Throughout the training process of the system, the agent is randomly sampled from the empirical playback pool with sampling points $t_{N_{bs}}$. The real Q-values obtained using the target critic network are as follows [30].

$$\begin{aligned} V_{\mu_i}(s_y'(t_{N_{bs}}) = r_i\Big(s_i(t_{N_{bs}}), a_i(t_{N_{bs}})\Big) + \\ \gamma \max_{a_i'(t_{N_{bs}})} Q_i\Big(s_i'(t_{N_{bs}}), a_i'(t_{N_{bs}})|\theta^{Q_i'}\Big) \end{aligned} \tag{40}$$

To make the output Q-value of the critic network closer to the real Q-values $y'(t_{N_{bs}})$ and guide the training of the actor, the mean squared error (MSE) as a loss function to measure the training loss is defined as follows.

$$L(\theta^{Q_i}) = \frac{1}{N_{bs}} \sum_{b=1}^{N_{bs}} \Big(y'(t_{N_{bs}}) - Q(s_i(t), a_i(t)|\theta^{Q_i})\Big)^2 \tag{41}$$

Using the gradient descent method, the critic network parameters θ^{Q_i} are optimized, which is denoted as

$$\theta^{Q_i} \leftarrow \theta^{Q_i} - \eta \nabla_{\theta^{Q^i}}(\theta^{Q^i}), \tag{42}$$

where η represents the parameter of the update step size. The objective of the actor Network aims to find the optimal policy that maximizes the Q value, which can be expressed as:

$$\mu_i(s_i|\theta^{\mu_i}) = \arg\max_{a_i} Q\Big(s_i(t), a_i(t)|\theta^{\mu_i}\Big). \tag{43}$$

The performance objective function for the current policy evaluation is designed as follows.

$$\mathcal{J}_\beta(\mu_i) = E_{s_i \sim \rho^\beta}[Q(s_i(t), a_i(t)|\Theta^{\mu_i})] \tag{44}$$

It estimates the expected value of the evaluation network $Q\Big(s_i(t), a_i(t)|\theta^{\mu_i}\Big)$ under the state distribution $s_i \sim \rho^\beta$. Then, the chain rule is applied to update the actor network, as follows.

$$\begin{aligned} \nabla_{\theta^{\mu_i}} \mathcal{J}_\beta(\mu_i) = E_{s_i \sim \rho^\beta}\Big[\nabla_{a_i} Q(s_i, a_i|\theta^{\mu_i})|_{a_i=\mu_i(s_i|\theta^{\mu_i})} \\ \cdot \nabla_{\theta^{\mu_i}} \mu_i(s_i|\theta^{\mu_i})\Big] \end{aligned} \tag{45}$$

The target parameters $\theta^{Q_i'}$ and $\theta^{\mu_i'}$ are updated using a soft update, as follows,

$$\theta^{Q_i'} = \theta^{Q_i} + (1-\tau)\theta^{Q_i'}, \tag{46}$$

$$\theta^{\mu_i'} = \theta^{\mu_i} + (1-\tau)\theta^{\mu_i'}, \tag{47}$$

where τ represents the discount factor. During model training, the critical network parameters are updated using Monte Carlo sampling (41), and the target network parameters are optimized using soft updates (46) and (47). The user-side cache decision process is shown in the algorithm 1.

Algorithm 1 AWD2PG: Local Cache Decision.

Initialize the replay memory Ω_i.
Initialize θ^{μ_i}, θ^{Q_i} and obtain $\theta^{\mu_i'}$, $\theta^{Q_i'}$ by cloning θ^{μ_i},θ^{Q_i}.
for $t = 0, 1, 2, 3, ..., T$ **do**
  for UE-$i \in \mathcal{N}$ in Parallel: **do**
    UE-i send requests at t by $q_i(t)$.
    Send missed files and user-related parameters to the server.
    Receive the Number of channels c_i (34) allocated to UE-i.
    Observer the state $s_i(t)$, and selects action by $a_i(t) = \mu_i(s_i(t)|\theta^{\mu_i})$.
    Observer reward feedback $r_i(t)$ (26), and obtain new observations $s_i'(t)$.
    Store $\{(s_i(t), a_i(t), r_i(t), s_i'(t))\}$ into its replay buffer $\mathbf{\Omega_i}$.
    Randomly select a mini-batch of transitions $\{(s_i(t), a_i(t), r_i(t), s_i'(t))\}$ from Ω_i.
    Calculate $y'(t_{N_{bs}})$ by (40), Then update θ^{Q_i} by (42).
    Soft-update the target actor/critic every φ steps as (46)(47).
  end for
end for

D. MEC Server Side Cache Decision

On the MEC server side, it receives all user cache decision-making actions and requests, which leads to a sharp increase in the dimension of the MEC side state, and it is impossible to create a Q table for all possible state/action pairs. Therefore, on the MEC server side, the TD3 algorithm is used to solve the joint optimization problem, as shown in fig. 4. The TD3 architecture consists of value networks Q_{θ_1} and Q_{θ_2}, and actor network π_ϕ. The online network corresponding to the critic network is $Q_{\theta_1'}$ and $Q_{\theta_2'}$, while the online network corresponding to the actor network is $\pi_{\phi'}$. Since the server receives proactive caching actions and lost files from all users in the coverage area, and both proactive caching and reactive caching files on the user side need to be transmitted through the server. Therefore, the server's caching decision-making scope includes the above two types. The proactive cache files

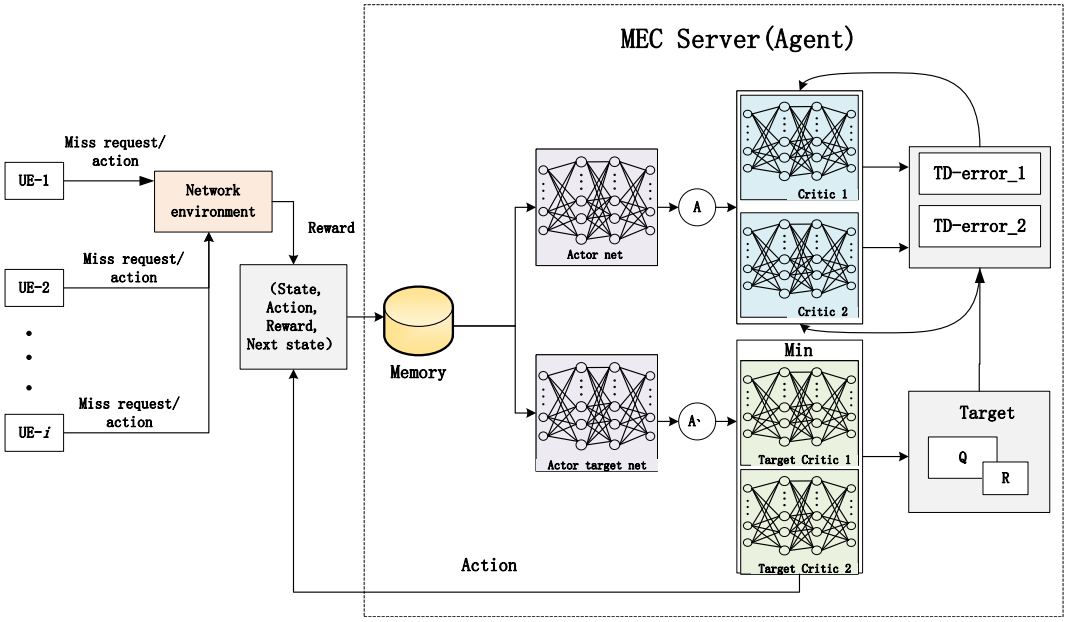

Fig. 4. The TD3 architecture applied to the MEC side.

of all users within the coverage area can be expressed as $PT(t) = \sum_{i=1}^{\mathcal{N}} PT_i(t)$ and $Q^G(t)$. Thus, the server's state $s_B(t)$ can be rewritten as,

$$\overline{s}_B(t) = \{CU_B(t), Q^G(t) \cup PT(t)\}. \tag{48}$$

The state $\overline{s}_B(t)$ is sent to the actor network and the state is mapped to a particular policy using $a_B(t) = \mu_B(\overline{s}_B(t)|\phi)$, which can be expressed as

$$a_B(t) = \mu_B(\overline{s}_B(t))|\phi) + n_B(t)|_{n_B(t)\sim N(0,\sigma^2)}, \tag{49}$$

where $n_B(t)$ is the noise vector that follows the mean distribution. After receiving action $a_B(t)$, the server updates its cache state $CU_B(t)$ to $CU_B(t+1)$ in response to the next server-side request. During the agent learning phase, a small amount of random noise is added to the target actions and averaged over a small batch to smooth out the target policy.

$$\begin{aligned}\overline{a}_B(\overline{s}_B(t)) = \mu_B(\overline{s}_B(t)|\phi') + clip\big((\epsilon, -c, c\big),\\ a_{Low}, a_{High}\Big), \epsilon \sim \mathcal{N}(0, \sigma)\end{aligned} \tag{50}$$

The action $\overline{a}_B(\overline{s}_B(t))$ is fed into the two critic target networks, and the Q-values corresponding to the current state and action are calculated as follows.

$$Q\Big(\overline{s}_B(t), \overline{a}_B(\overline{s}_B(t))|\theta_1\Big) = E_{\mu_B}\Big[\sum_{\tau=0}^{+\infty} \chi^\tau r_B(t+\tau)|\overline{s}_B(t), \overline{a}_B(t)\Big] \tag{51}$$

$$Q\Big(\overline{s}_B(t), \overline{a}_B(\overline{s}_B(t))|\theta_2\Big) = E_{\mu_B}\Big[\sum_{\tau=0}^{+\infty} \chi^\tau r_B(t+\tau)|\overline{s}_B(t), \overline{a}_B(t)\Big] \tag{52}$$

The Clipped Double Q-learning method is used to learn the two Q functions independently, and the smaller Q-value is used for network updates [31].

$$y\Big(\overline{s}_B(t), \overline{a}_B(\overline{s}_B(t))\Big) = r_B + \gamma \min_{i=1,2}\Big(Q(\overline{s}_B(t), \overline{a}_B(\overline{s}_B(t))\Big|\theta_i'\Big). \tag{53}$$

The predicted values are fed back to the critic network, and the parameters of the two critic networks are updated separately.

$$L(\theta_1) = \Big[Q(\overline{s}_B(t), \overline{a}_B(\overline{s}_B(t))|\theta_1) - y\Big(\overline{s}_B(t), \overline{a}_B(\overline{s}_B(t))\Big)\Big]^2 \tag{54}$$

$$L(\theta_2) = \Big[Q(\overline{s}_B(t), a_B(\overline{s}_B(t))|\theta_2) - y\Big(\overline{s}_B(t), \overline{a}_B(\overline{s}_B(t))\Big)\Big]^2 \tag{55}$$

By maximizing the cumulative expected return, the parameters of the two critic networks are updated. The deterministic strategy gradient algorithm is used by the actor network to generate optimal strategies by maximizing the Q-value and updating the strategy network parameters.

$$\begin{aligned}\nabla_\phi \mathcal{J}_\beta(\phi) = E_{\mu_B}[\nabla_{a_B} Q(s_B, a_B|\phi)|_{a_B=\mu_B(s_B|\theta^{\mu_B})}\\ \cdot \nabla_\phi \mu_B(s_B|\phi)]\end{aligned} \tag{56}$$

MEC server-side cache decision is outlined in Algorithm 2, where δ is the total episodes.

Algorithm 2 AWD2PG: MEC Server Side Cache Decision.

Initialize the replay memory Ω_B.
Initialize critic networks Q_{θ_1} and Q_{θ_2}, and the actor network with random parameters ϕ.
Initialize target networks $\theta_2' \leftarrow \theta_2$, $\theta_1' \leftarrow \theta_1$, $\phi' \leftarrow \phi$..
for episode$= 1, 2, 3, ..., \delta$ **do**
  for $t = 0, 1, 2, 3, ..., T$ **do**
    UE-i send requests at t by $q_i(t)$.
    Receive all the user absent files and user-related parameters.
    The channel assignment weight is calculated by the attention mechanism (36), and the number of channels c_i(34) is sent to UE-i.
    Local cache update phase.
    Receive proactive cache action $PT(t)$ and missing files $Q^G(t)$ from users.
    Observe the state $\overline{s}_B(t)$ and select the action $a_B(t)$ with exploration noise by (49), receive reward feedback $r_B(t)$, and obtain new observations $s_B'(t)$.
    Store $\{\overline{s}_B(t), \overline{a}_B(t), r_B(t), s_B'(t)\}$ into its replay buffer Ω_B.
    Randomly select a mini-batch of transitions from Ω_B.

    Calculate $\overline{a}_B$ by (50), and Calculate y by (53); Then update the critic network parameters θ_1 and θ_2 by (54) and (55), respectively.
    Update the actor network parameter ϕ by (56).
    Update target networks:
  end for
end for

E. Convergence and Complexity

The time computational complexity and convergence of the proposed algorithm are analyzed according to [32].

User side cache algorithm: As mentioned above, we define the number of users as N, the number of contents as F, the cache capacity of users as M_i, and the number of channels as C. In DDPG, the two phases that require high computing power are the training of actor network and critic network. The time calculation complexity of actor network is $O(D_{A_i}D_{A_u} + \sum_{u=1}^{U-2} D_{A_u}D_{A_{u+1}} + D_{A_i}D_{A_u})$, where $D_{A_i} = S_i$, representing the input layer. The number of neurons in the output layer is $D_{A_u} = C + M_i$. Therefore, the computational complexity of the actor network is $O(\gamma_a = \delta_{MB}T_1(D_{A_i}D_{A_u} + \sum_{u=1}^{U-2} D_{A_u}D_{A_{u+1}} + D_{A_i}D_{A_u}))$. where δ_{MB} represents the batch size and T_1 is the upper bound of the training steps. The time computational complexity of the critic network is $O(\gamma_b = \delta_{MB}T_1(D_{C_i}D_{C_u} + \sum_{u=1}^{U-2} D_{C_u}D_{C_{u+1}} + D_{C_i}D_{C_u}))$, where the number of neurons in the input layer is $D_{C_i} = C + M_i + S$, and the number of neurons in the output layer is $D_{C_u} = 1$. Therefore, the time complexity of the proposed DDPG algorithm is $O(max(\gamma_a, \gamma_b))$.

Server side cache algorithm: Similar to the user-side algorithm, the time computational complexity of the actor network in the server-side TD3 algorithm can be expressed as $O(\gamma_a = \delta_{MB}T_1(E_{A_i}E_{A_u} + \sum_{u=1}^{U-2} E_{A_u}E_{A_{u+1}} + E_{A_i}E_{A_u}))$, where $E_{A_i} = S_B$ represents the number of neurons in the input layer, and E_{A_u} represents the neurons in the output layer quantity. The critic network in the TD3 algorithm also needs to be updated. If we assume that the structure of the double Q network in the TD3 algorithm is the same as the critic network in DDPG, then the time computational complexity can be expressed as $O(2\gamma_b = 2\delta_{MB}T_1(E_{C_i}E_{C_u} + \sum_{u=1}^{U-2} E_{C_u}E_{C_{u+1}} + E_{C_i}E_{C_u}))$, where E_{C_i} represents the number of neurons in the input layer, and E_{C_u} represents The number of neurons in the output layer. Therefore, the update of the TD3 algorithm involves updating the parameters of the actor network and the critic network, and the time complexity of the update step can be expressed as $O(max(\gamma_a, 2\gamma_b))$.

Proposed AWD2PG algorithm: During each iteration, two subproblem algorithms are executed to solve the above two subproblems. The complexity of the user-side and server algorithms has been analyzed above. Therefore, the complexity of the proposed AWD2PG algorithm is $O(max(\gamma_a, 2\gamma_b))$.

VI. NUMERICAL SIMULATION AND ANALYSIS

In this section, to evaluate the performance of the proposed AWD2PG method, the proposed algorithm is compared with other algorithms under different parameters.

A. Parameter Setting

In the simulation, the total number of files was set by default to $F = 48$. Because the network adopts fragmented transmission, all files were considered equivalent in size [33]. Furthermore, the cache size was equal for all users, i.e., $m_i = m_j, \forall \in i, j \in \mathcal{N}, i \neq j$. The cache capacity of users and MEC servers were respectively set to 4 and 8 by default, and it was assumed that the cloud server could store all content. According to [34], the popularity distribution of BBC iPlayer video files requested by users in June 2014 can be approximated by a Zipf distribution with parameter $(\tau, \beta) = (-0.86, 0.52)$. In addition, each user UE and BS locally runs a DRL agent with a three-layer neural network, all of which models adaptively learn their respective training parameters using the Adam optimizer, starting from 10^{-4}. In addition, we set the batch size to $[128, 256]$ to represent the heterogeneity of the user model, the learning rate of the model is set to 0.01, and the reward decay is set to 0.8. For the DRL agent at the MEC server, the noise $n_B(t)$ of its action is set to 0.5. The simulation parameter settings of DRL are shown in Table II.

TABLE II
MODEL SIMULATION PARAMETERS

Parameter	Value
learning rate	0.01
policy noise	0.5
action dim	300
batch size	256
gamma	0.8
Exploration rate	0.1
Hidden layer	3
Activation function	ReLU

We set the time slot of the system to 250, and the number of user requests in each time slot follows a Poisson distribution

of $lam = 0.5$. In each time slot, the server uses the attention mechanism to allocate a certain number of wireless channels to all users within the coverage area according to the received information of all users. The attention mechanism used in the experiment is Scaled dot-product attention, mainly by mapping the input sequence and then calculating the dot product, then scaling the dot product result, and finally weighting the scaled result with the input vector , so as to obtain the output of the self-attention mechanism.

B. Baseline Schemes and Performance Metrics

To evaluate the proposed AWD2PG model, some baseline caching schemes are used for comparison such as follows.

- **Least Recently Used (LRU)**: The LRU scheme is replacing the cached file that has not been used for the longest time when a new file arrives.
- **Least Frequently Used (LFU)**: The LFU scheme replaces the data with the lowest usage frequency, and if there are other data with the same usage frequency, the data with the longest unused time is evict.
- **First-In-First-Out (FIFO)**: The FIFO scheme is replacing the file that has been in the cache for the longest time.
- **Random choice (RC)**: The RC algorithm randomly selects the content to be cached and replaced.
- **Most popular (MP)** [32]: The MP caching scheme utilizes prior knowledge of user request patterns and the M-Zipf request popularity model to estimate the likelihood of file requests as the number of requests increases. The file with the highest probability of being requested is then selected for caching, aligning caching decisions with user request popularity.

The effectiveness of the proposed AWD2PG algorithm was evaluated in the following aspects. Firstly, the convergence of the algorithm was verified. Secondly, the proposed caching mechanism was evaluated based on the cache hit ratio and network load, both on the user side and the server side. The cache hit ratio on the user side was measured as the number of hits divided by the total number of user requests, while the network load included proactive transmission and the number of missing files requested. On the server side, the evaluation considered the cache hit rate and the number of proactively transmitted and missing files over the backhaul link.

C. Algorithm Evaluation

The comparison between the proposed AWD2PG algorithm and N-AWD2PG algorithm in terms of convergence behavior is depicted in Fig. 5. In this context, N-AWD2PG refers to the method of sharing a certain number of channels on average among all the users within the range of the base station during file transmission. The figure illustrates the reward value of the proposed AWD2PG algorithm compared to N-AWD2PG algorithm, with the total channel capacity set to $C(t) = 28$. It is evident that the AWD2PG algorithm achieves a higher reward value faster and more smoothly than N-AWD2PG algorithm due to its ability to dynamically allocate the number of channels based on the local model and the number of requests. The suboptimal performance of N-AWD2PG is mainly attributed to two situations: one is the poor quality of the decision model maintained by the user, and the other is when the number of files requested by the user within a given time slot is small. Both of the above will allocate too many channels to users, which leads to the proactive transmission of files being unable to meet user requests for the next time slot, thereby increasing network traffic load.

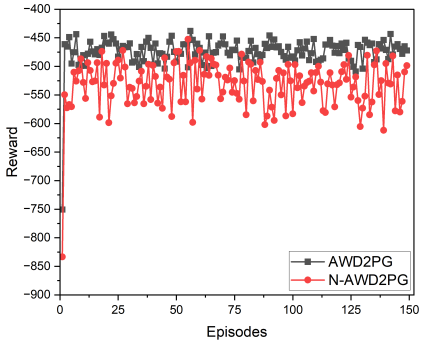

Fig. 5. The demonstration of the reward of the AWD2PG and N-AWD2PG algorithms.

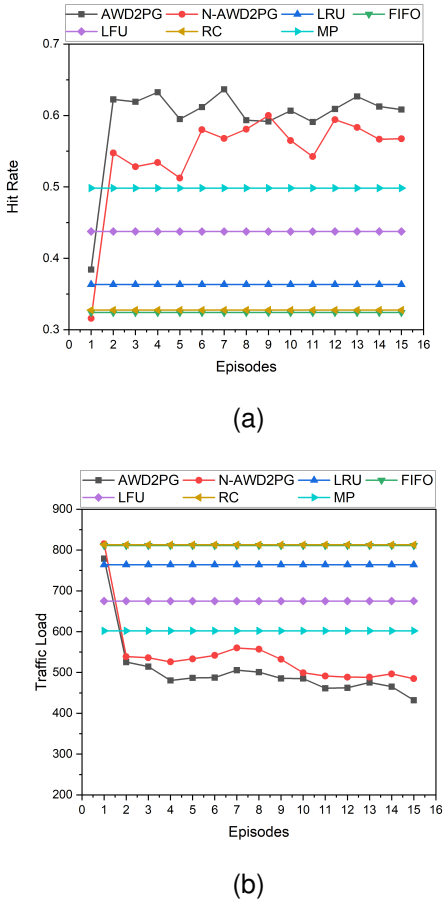

Fig. 6. The performance evaluation of different schemes in terms of the (a) hit rate and (b) traffic load with respect to time

Fig. 6 demonstrates the change of network load and hit rate

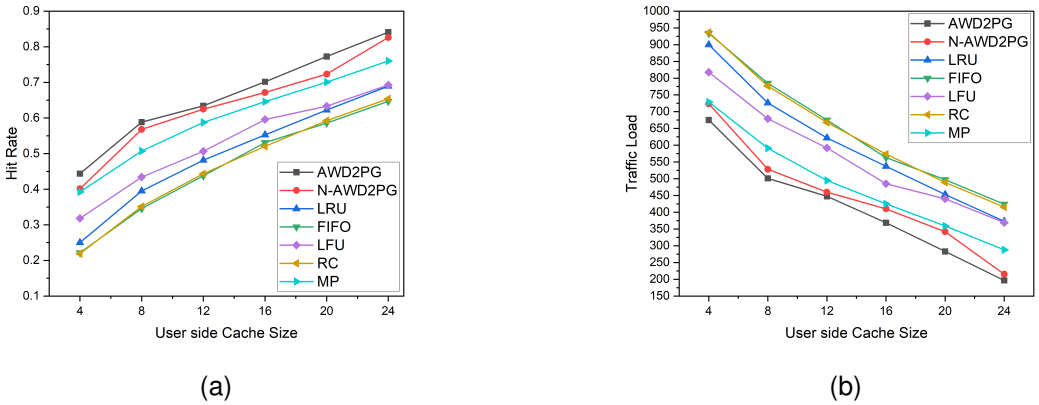

Fig. 7. The performance evaluation of different cache capacities on the user side in terms of the (a) hit rate and (b) network traffic load.

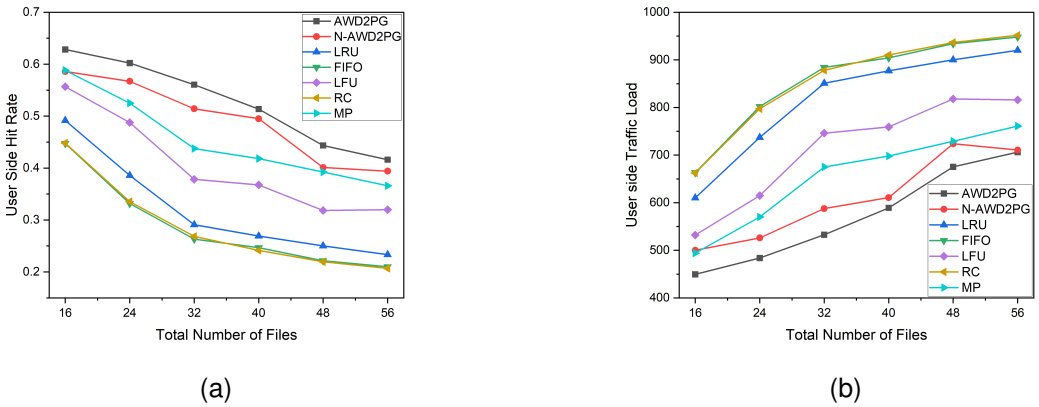

Fig. 8. The performance evaluation of the models with different total numbers of files on the user side in terms of the (a) hit rate and (b)network traffic load.

during the training process of the proposed strategy, where the total number of parameter files $\mathcal{N} = 48$, and the user's cache size $m_i = 8$. It can be seen from the figure that the proposed AWD2PG strategy achieves the lowest network load and the largest cache hit ratio, and is always better than other strategies. In terms of hit rate, the proposed AWD2PG algorithm outperforms N-AWD2PG, MP, LFU, LRU, FIFO and RC algorithms by 10.17%, 19.61%, 36.24%, 64.05%, 82 % and 83.87%. According to Fig. 6(b), the proposed AWD2PG scheme outperforms the first six baseline schemes by 5.66%-37.47% in terms of network traffic.

D. Performance Evaluation of User-side Caching Decisions

In this section, simulation results are presented to validate the performance of the proposed AWD2PG framework on the user side. Given that the MP algorithm is a local caching strategy based on the user request model, a preprocessing approach can be used to obtain prior knowledge of the likelihood of the requested file. Fig. 7 and Fig. 8 show how the proposed dynamic caching policy AWD2PG compared to N-AWD2PG and the baseline on the user side in terms of two factors, namely the network transfer load cost, and cache hit rate, which are the local cache size M_i and the total number of files F, respectively. As displayed in the figures, the proposed caching policy AWD2PG outperformed the others. Fig. 7 demonstrates that the proposed strategy achieved a low network load and a high cache hit ratio for various cache capacities. This implies that the proposed approach is more competitive in terms of cache optimization, especially when user cache resources are limited. This improvement is primarily due to the ability of the proposed algorithm to more accurately predict the user request transition probability and update the cached content.

Fig. 7(a) depicts the effect of the user cache size m_i on the cache hit rate. As the cache size increases, the cache has a greater probability of storing the requested file, and thus its cache hit rate increases. Specifically, compared to the N-AWD2PG, LRU, FIFO, LFU, RC, and MP algorithms, the proposed AWD2PG algorithm exhibited average increases in the hit rate of 4.92%, 33.11%, 43.81%, 25.16%, 43.17%, and 10.77%, respectively.

Fig. 7(b) depicts the effect of the cache capacity on the network traffic load. The network traffic load was found to decrease monotonically with the monotonic growth of the cache, for when the cache capacity is large enough, the transmission of request traffic caused by file misses is reduced. For

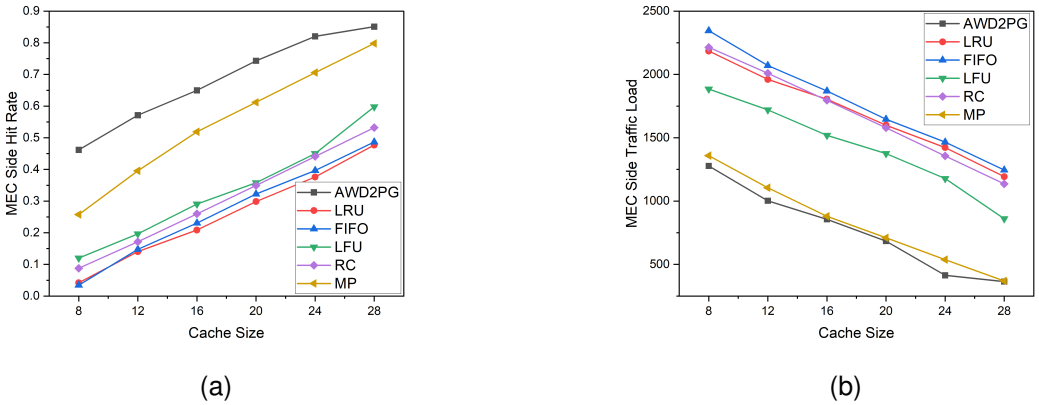

Fig. 9. The impact of the cache size on the MEC server side in terms of the (a) hit rate and (b) network traffic load.

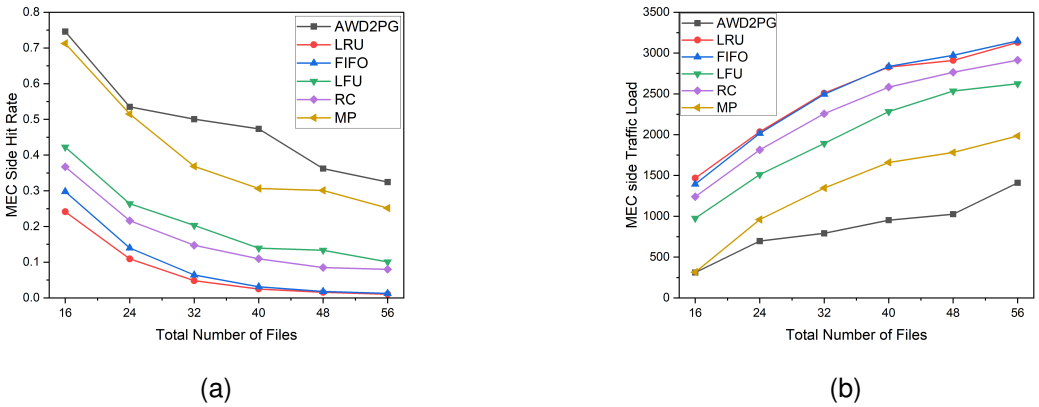

Fig. 10. The impact of the number of total contents F at the MEC server side with M_i = 4 and M_B = 8 in terms of the (a) hit rate and (b) network traffic load.

various cache capacities, the proposed AWD2PG algorithm outperformed the N-AWD2PG, LRU, FIFO, LFU, RC, and MP algorithms by 8.53%, 31.54%, 36.25%, 26.93%, and 14.37%, respectively.

The performance of the models was compared when the total number of files F varied from 16 to 64 and the user cache capacity was $m_i = 4$. In terms of the network load and cache hit ratio, Fig. 8 shows that the proposed AWD2PG algorithm outperformed all baseline methods. It can also be seen that as the total number of files increased, the accuracy of the caching algorithm decreased, and the cache performance gradually decreased. Fig. 8(a) depicts the effect of the total number of files on the cache hit rate. The cache hit rate appeared to decrease monotonically as the total number of files increased. The proposed AWD2PG and MP algorithms had a performance difference of 6.77% for a total of 16 files and 13.74% for a total of 56 files. Compared to the N-AWD2PG, LRU, FIFO, LFU, and RC algorithms, the performance increased by 6.99%, 64.72%, 83.87%, 30.3%, 83.94%, and 16.01%, respectively.

Fig. 8(b) shows the effect of the total number of files on the network traffic load. The network load of the proposed AWD2PG algorithm was lower than that of the baseline method and increased along with the total number of files.

E. Performance Evaluation of MEC-side Caching Decisions

Considering that the MP algorithm is a local caching policy based on the user request model, and the server receives missing files from all users in the coverage area, the request model on the BS side cannot be obtained directly. Therefore, the MP algorithm was implemented during pre-processing to generate a global popularity model to predict the request probability of each file on the BS side. On the MEC server side, the proposed AWD2PG algorithm compares the network transmission load cost and cache hit rate with all baselines in terms of cache size M_B and the total number of files F, respectively. As shown in Fig. 9, AWD2PG exhibited the same performance variation trend on the server side as on the user side under different factors, and outperformed the other baseline methods on both the server and user sides.

Fig. 9 compares the performance of various caching schemes on the server side at various cache capacities, ranging from 8 to 28. Fig. 9 (a) depicts the relationship between the hit rate and the cache capacity. The proposed AWD2PG algorithm outperformed all baseline schemes, with a 24.6%-165.56% average hit rate improvement and a 6.66%-79.37% improvement over the MP algorithm. Fig. 9 (b) shows that the

proposed AWD2PG scheme achieved a lower network load than all baseline schemes. Fig. 9 also shows that AWD2PG consistently outperformed the MP algorithm with a priori knowledge for all factors. Furthermore, with the increase of the cache size, the gap between AWD2PG and MP shrunk, because as the cache size grows, there will be more cache capacity to store files with a high request probability.

The impact of the total number of files on server-side performance was subsequently investigated in greater depth, where the total number of files F ranges from 16 to 64. Fig. 10 depicts the impact of the total number of files on the server-side cache performance. Fig. 10(a) shows that on the server side, the proposed AWD2PG algorithm consistently outperformed the baseline algorithms in terms of the hit rate. Fig. 10(b) shows that the proposed AWD2PG algorithm offloaded more backhaul link traffic and reduced it by averages of 65.14%, 65.10%, 56.13%, 61.78%, and 35.55% compared to LRU, FIFO, LFU, RC, and MP algorithms, respectively.

VII. CONCLUSION

In this study, the optimization problem of joint proactive caching and cache replacement for edge caches in mobile networks is investigated. Furthermore, the joint optimization problem at the user side and edge side was respectively formulated as an infinite-range average cost MDP process with a cost function defined to minimize the network load. The AWD2PG framework is proposed to solve the joint cache optimization problem on the user and server sides separately while controlling the allocation of limited available channels between them. In particular, the DDPG and TD3 models are adopted on the user side and the server side respectively to deal with the caching decision problem based on network status and historical data. Most importantly, an attention mechanism is introduced to control the allocation weights of the limited available channels on the user and server side to solve uneven channel allocation among users. Simulation results show that the proposed strategy is effective in predicting future user requests and outperforms existing schemes compared to baselines. In future research, the focus will be more on practical considerations, exploring request distribution models for different data types, and edge caching models for D2D and cloud-edge device collaboration.

REFERENCES

[1] Wang X, Li X, Pack S, *et al*. STCS: Spatial-temporal collaborative sampling in flow-aware Software defined networks[J]. *IEEE journal on selected areas in communications*, 2020, 38(6): 999-1013.

[2] Fang J, Ma A. Iot application modules placement and dynamic task processing in edge-cloud computing[J]. *IEEE Internet of Things Journal*, 2020, 8(16): 12771-12781.

[3] Podlipnig S , Boeszoermenyi L . A survey of Web cache replacement strategies[J]. *Acm Computing Surveys*, 2003, 35(4):374-398.

[4] Hachem J, Karamchandani N, Diggavi S. Content caching and delivery over heterogeneous wireless networks[C]//*2015 IEEE conference on computer communications (INFOCOM). IEEE*, 2015: 756-764.

[5] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch and G. Caire, "FemtoCaching: Wireless Content Delivery Through Distributed Caching Helpers," in *IEEE Transactions on Information Theory*, vol. 59, no. 12, pp. 8402-8413, Dec. 2013, doi: 10.1109/TIT.2013.2281606.

[6] Liu S, Zheng C, Huang Y, *et al*. Distributed Reinforcement Learning for Privacy-Preserving Dynamic Edge Caching[J]. *IEEE Journal on Selected Areas in Communications*, 2022, 40(3): 749-760.

[7] Yan S, Jiao M, Zhou Y, *et al*. Machine-learning approach for user association and content placement in fog radio access networks[J]. *IEEE Internet of Things Journal*, 2020, 7(10): 9413-9425.

[8] Tadrous J, Eryilmaz A. On optimal proactive caching for mobile networks with demand uncertainties[J]. *IEEE/ACM Transactions on Networking*, 2015, 24(5): 2715-2727.

[9] Qian Y , Wang R , Wu J, *et al*. Reinforcement Learning Based Optimal Computing and Caching in Mobile Edge Network[J]. *IEEE Journal on Selected Areas in Communications*, 2020, PP(99):1-1.

[10] Somuyiwa S O, György A, Gündüz D. A reinforcement-learning approach to proactive caching in wireless networks[J]. *IEEE Journal on Selected Areas in Communications*, 2018, 36(6): 1331-1344.

[11] Zheng Z, Song L, Han Z, *et al*. A stackelberg game approach to proactive caching in large-scale mobile edge networks[J]. *IEEE Transactions on Wireless Communications*, 2018, 17(8): 5198-5211.

[12] Sun Y, Cui Y, Liu H. Joint pushing and caching for bandwidth utilization maximization in wireless networks[J]. IEEE Transactions on Communications, 2018, 67(1): 391-404.

[13] Chen Q, Wang W, Chen W, *et al*. Cache-enabled multicast content pushing with structured deep learning[J]. *IEEE Journal on Selected Areas in Communications*, 2021, 39(7): 2135-2149.

[14] Wang F, Wang F, Liu J, *et al*. Intelligent video caching at network edge: A multi-agent deep reinforcement learning approach[C]// *IEEE INFOCOM 2020-IEEE Conference on Computer Communications. IEEE*, 2020: 2499-2508.

[15] Peng H, Shen X. Multi-agent reinforcement learning based resource management in MEC-and UAV-assisted vehicular networks[J]. *IEEE Journal on Selected Areas in Communications*, 2020, 39(1): 131-141.

[16] Alqahtani F, Al-Maitah M, Elshakankiry O. A proactive caching and offloading technique using machine learning for mobile edge computing users[J]. *Computer Communications*, 2022, 181: 224-235.

[17] Zhang Y, Li Y, Wang R, *et al*. PSAC: Proactive sequence-aware content caching via deep learning at the network edge[J]. *IEEE Transactions on Network Science and Engineering*, 2020, 7(4): 2145-2154.

[18] Yu Z, Hu J, Min G, *et al*. Privacy-preserving federated deep learning for cooperative hierarchical caching in fog computing[J]. *IEEE Internet of Things Journal*, 2021.

[19] Y. Lu, W. Chen and H. V. Poor, "Coded Joint Pushing and Caching With Asynchronous User Requests," in IEEE Journal on Selected Areas in Communications, vol. 36, no. 8, pp. 1843-1856, Aug. 2018, doi: 10.1109/JSAC.2018.2844918.

[20] W. Chen and H. V. Poor, "Content Pushing With Request Delay Information," in IEEE Transactions on Communications, vol. 65, no. 3, pp. 1146-1161, March 2017, doi: 10.1109/TCOMM.2017.2648800.

[21] Gregori M, Gómez-Vilardebó J, Matamoros J, et al. Wireless content caching for small cell and D2D networks[J]. IEEE Journal on Selected Areas in Communications, 2016, 34(5): 1222-1234.

[22] Somuyiwa S O, Gündüz D, Gyorgy A. Reinforcement learning for proactive caching of contents with different demand probabilities[C]// *2018 15th International Symposium on Wireless Communication Systems (ISWCS)*. IEEE, 2018: 1-6.

[23] Somuyiwa S O, György A, Gündüz D. Multicast-aware proactive caching in wireless networks with deep reinforcement learning[C]//2019 *IEEE 20th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. IEEE, 2019: 1-5.

[24] Gao S, Dong P, Pan Z, *et al*. Reinforcement learning based cooperative coded caching under dynamic popularities in ultra-dense networks[J]. *IEEE Transactions on Vehicular Technology*, 2020, 69(5): 5442-5456.

[25] Sadeghi A, Sheikholeslami F, Marques A G, *et al*. Reinforcement learning for adaptive caching with dynamic storage pricing[J]. *IEEE Journal on Selected Areas in Communications*, 2019, 37(10): 2267-2281.

[26] Sakr H, Elsabrouty M. Meta-reinforcement learning for edge caching in vehicular networks[J]. Journal of Ambient Intelligence and Humanized Computing, 2023, 14(4): 4607-4619.

[27] X. Zhou, Z. Ke and T. Qiu, "Recommendation-Driven Multi-Cell Cooperative Caching: A Multi-Agent Reinforcement Learning Approach," in IEEE Transactions on Mobile Computing, doi: 10.1109/TMC.2023.3297213.

[28] X. Wang, R. Li, C. Wang, X. Li, T. Taleb and V. C. M. Leung, "Attention-Weighted Federated Deep Reinforcement Learning for Device-to-Device Assisted Heterogeneous Collaborative Edge Caching," in *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 1, pp. 154-169, Jan. 2021, doi: 10.1109/JSAC.2020.3036946.

[29] Wang X, Wang C, Li X, *et al*. Federated deep reinforcement learning for Internet of Things with decentralized cooperative edge caching[J]. *IEEE Internet of Things Journal*, 2020, 7(10): 9441-9455.

[30] Lillicrap T P, Hunt J J, Pritzel A, *et al*. Continuous control with deep reinforcement learning[J]. arXiv preprint arXiv:1509.02971, 2015.
[31] Fujimoto S , Hoof H V , Meger D . Addressing Function Approximation Error in Actor-Critic Methods[J]. 2018.
[32] T. Zhang, Y. Wang, W. Yi, Y. Liu, C. Feng and A. Nallanathan, "Two Time-Scale Caching Placement and User Association in Dynamic Cellular Networks," in IEEE Transactions on Communications, vol. 70, no. 4, pp. 2561-2574, April 2022, doi: 10.1109/TCOMM.2022.3152265.
[33] Poularakis K, Iosifidis G, Sourlas V, *et al*. Multicast-aware caching for small cell networks[C]//2014 *IEEE wireless communications and networking conference [2CNC)*. IEEE, 2014: 2300-2305.
[34] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, "Measurement, modeling, and analysis of a peer-topeer file-sharing workload," in Proc. 19th ACM Symp. Operating Syst. Princ., 2003, pp. 314–329.

Juan Fang received the M.S. degree from Jilin University of Technology, Changchun, China in 1997 and her Ph.D. degree from the College of Computer Science, Beijing University of Technology, Beijing, China in 2005. In 1997, she joined the College of Computer Science, Beijing University of Technology. From 2015, she is the professor of Beijing University of Technology. She currently works with the Faculty of Information Technology, Beijing University of Technology, Beijing. Her research interests include high performance computing, heterogeneous intelligent computing and edge computing.

Huijing Yang received the B.S. degree from Zhoukou Normal University, Zhoukou, China in 2018. She is currently a Ph.D. student at Beijing University of Technology, Beijing, China. She is a student member of CCF. Her main research direction is computer architecture.

Yu Lu received the Msc degree in college of computer science from Beijing University of Technology in 2015. She is currently working towards a Ph.D. degree in computer science from James Cook University. Her research interests are computer vision and explainable AI.

Ziyi Teng received the B.S. degree from the North China University of Water Resources and Electric Power, Zhengzhou, China in 2019. She is currently pursuing the Ph.D. degree with the Beijing University of Technology, Beijing, China. She is a student member of CCF. Her research interests in mobile edge computing.

Huijie Chen received the B.Eng degree from the School of Computer Science, Henan University of Economics and Law, Zhengzhou, China in 2010, the M.Seng degree from the School of Computer Science, Taiyuan University of Science and Technology, Taiyuan, China in 2013, and Ph.D. degree in computer sience from the Beijing Institute of Technology, Beijing, China in 2020. He currently works in the school of computer science, Beijing University of Technology, Beijing, China. His research interests include Smart Sensing, crowdsensing, and mobile edge computing.

Wei Xiang (Senior Member, IEEE) received the B.Eng. and M.Eng. degrees in electronic engineering from the University of Electronic Science and Technology of China, Chengdu, China, in 1997 and 2000, respectively, and the Ph.D. degree in telecommunications engineering from the University of South Australia, Adelaide, SA, Australia, in 2004. He is the Cisco Research Chair of AI and IoT and Director Cisco-La Trobe Centre for AI and IoT, the School of Engineering and Mathematical Sciences, La Trobe University, Melbourne, VIC, Australia. He is currently also a part-time research fellow with the Network Communication Research Center, Peng Cheng Laboratory, Shenzhen, China. He was the Foundation Chair and the Head of Discipline of IoT Engineering, James Cook University, Cairns, QLD, Australia. Due to his instrumental leadership in establishing Australia's first accredited Internet of Things Engineering degree program, he was inducted into Pearcy Foundation's Hall of Fame in October 2018. He has published over 250 peer-reviewed papers, including three books and 180 journal articles. His research interest includes the Internet of Things, wireless communications, machine learning for IoT data analytics, and computer vision.