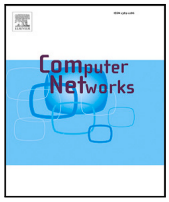
Computer Networks 242 (2024) 110247

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



Cooperative caching algorithm for mobile edge networks based on multi-agent meta reinforcement learning

Zhenchun Wei [a,c,d], Yang Zhao [a], Zengwei Lyu [a,c,d,*], Xiaohui Yuan [b], Yu Zhang [a], Lin Feng [a]

[a] School of Computer Science and Information Engineering, Hefei University of Technology, 193 Tunxi Road, Hefei, 230009, Anhui, China
[b] Computer Science and Engineering, University of North Texas, 1155 Union Circle, Denton, 76210, TX, USA
[c] Anhui Province Key Laboratory of Industry Safety and Emergency Technology, 193 Tunxi Road, Hefei, 230009, Anhui, China
[d] Engineering Research Center of Safety Critical Industrial Measurement and Control Technology, Ministry of Education, 193 Tunxi Road, Hefei, 230009, Anhui, China

ARTICLE INFO

Keywords:
Mobile edge computing (MEC)
Cooperative caching
Meta reinforcement learning (MRL)
Multi-agent system

ABSTRACT

Edge caching reduces service latency, relieves backhaul link traffic pressure, and enhances the quality of experience by multiplexing network content. Many studies use deep reinforcement learning (DRL) methods to develop edge caching policies. However, these traditional DRL methods have limitations, such as lengthy training time, poor model generalization, and the need to relearn network parameters for new tasks. To overcome these challenges, this paper proposes a multi-agent meta reinforcement learning-based cooperative edge caching algorithm (MAMRC), which consists of an inner and an outer model. The inner model employs the multi-agent deep reinforcement learning (MADRL) algorithm to implement cooperative caching for distributed base stations (BSs). It improves the cache hit rate and reduces service latency. The outer model uses a meta-learning method to learn meta-parameters and initialize the inner model, which enhances the generalization capability of the inner model, allowing it to rapidly adapt to new tasks. Experiment results indicate that, compared to the traditional DRL method and the MADRL-based algorithm, the inner model caching performance considering edge collaboration is improved by 15.35% and 4.55% respectively. Notably, compared to traditional caching algorithms and the inner model without initialization with meta-parameters, MAMRC demonstrates superior average caching performance and stronger generalization ability when facing new tasks.

1. Introduction

The growing popularity of smart devices has led to a significant rise in mobile data traffic, which is expected to continue increasing in the foreseeable future [1]. According to Cisco's report [2], the number of internet users will reach 5.3 billion, and the number of connected devices will reach 29.3 billion by 2023. To meet the soaring demand for data traffic, MEC has emerged as a critical technology to address the challenges posed by the densification of base station deployments in next-generation networks [3]. However, the deployment of BSs alone is not enough to solve the problems of heavy backhaul link burden and high content delivery latency, which arise due to a large number of repeated requests for popular content from mobile users [4]. On the other hand, edge caching can effectively cache frequently accessed content closer to the user at the edge server, enabling many repetitive content requests to be satisfied directly by the edge caching [5]. Edge caching significantly reduces the burden on the core network, resulting in improved quality of experience (QoE) [6]. The outstanding performance of edge caching in enhancing network quality of service has attracted the attention of numerous scholars, and it is recognized as one of the current hot research topics in the field of MEC by the European Telecommunications Standards Institute [7].

Early work related to edge caching utilized previously known content popularity to design caching policies [8,9]. However, these approaches relied on assumptions that are unsuitable for realistic scenarios where content popularity is often unavailable [10]. To address this issue, studies have independently designed caching mechanisms for each edge node, considering the differences among BSs [11–13]. However, when BSs with limited caching capacity make independent decisions, they may cache popular content repeatedly, resulting in the underutilization of caching resources [14]. With the rapid development of MADRL, cooperative caching mechanisms in MEC scenarios have received increased research attention [15]. By sharing

* Corresponding author.
E-mail address: lzw@hfut.edu.cn (Z. Lyu).

https://doi.org/10.1016/j.comnet.2024.110247
Received 4 June 2023; Received in revised form 24 November 2023; Accepted 12 February 2024
Available online 14 February 2024

cache resources, BSs can mutually cooperate to avoid caching excessive redundant data [16]. Nevertheless, traditional DRL methods used to develop caching strategies have certain limitations, including low sample utilization, long training time, and poor model generalization ability [17].

In a cooperative caching system where BSs have limited cache capacity, the caching decisions made independently by each BS may lead to redundant caching of popular content and reduced cache utilization. Moreover, traditional DRL algorithms face challenges adapting to new tasks, resulting in increased content delivery latency and decreased QoE [18]. Developing an edge cooperative caching policy with strong generalization capability is thus a pressing challenge. First, traditional optimization decision algorithms are typically designed to optimize short-term objectives, which may not be adequate to handle the non-linear, non-stationary, and non-deterministic MEC network and the long-term payoff problem. Second, the caching decision of a single BS in cooperative caching is influenced by other BSs' cache state and cache actions. However, when BS performs cache replacement actions, it can only observe its local status and not the cache actions of others. Third, traditional DRL algorithms have poor robustness and generalization performance, and their performance is typically limited to training tasks, requiring retraining when facing new tasks [19]. For instance, a new content request popularity distribution can lead to the failure of the trained caching model's parameters, leading to repeated learning from scratch, which incurs significant time and data cost consumption [20].

This paper presents a cooperative edge caching algorithm that introduces a meta-learning approach [21] into MADRL. MAMRC learns an adaptation to the training task, which consists of an inner and an outer model. The inner model learns the cooperative edge caching strategy using the meta-parameters provided by the outer model and evaluates the quality of the meta-parameters, providing feedback to the outer model. The outer model learns the meta-parameters using the training task set and adjusts them based on the feedback from the inner model. The outer model uses the meta-knowledge accumulated from the relevant tasks to guide the inner model, enabling it to adapt quickly to new tasks. The main contributions of this paper include

- A distributed cooperative edge caching algorithm is designed to maximize the long-term caching gain of the MEC network by designing the state space, action space, and reward function to optimize the collaboration mechanism among BSs.
- A MRL-based initial parameter training algorithm is proposed that guides subsequent learning by pre-storing helpful shared experience from the training task in advance, improving the generalization ability of the caching decision algorithm when facing new tasks.

The rest of this paper is organized as follows: Section 2 provides an overview of related work. Section 3 introduces the system model and elaborates on the proposed problem. Section 4 defines a deep reinforcement learning model for solving the optimal edge cooperative caching policy. In Section 5, the structure and network parameter update details of the proposed algorithm MAMRC are discussed. Section 6 presents simulation experiments and discusses the algorithm's performance. Finally, Section 7 concludes the paper and provides an outlook on future work.

2. Related work

The traditional content replacement methods, including least frequently used (LFU), least recently used (LRU), and first-in first-out (FIFO) [22], have been used widely. However, these methods may not be effective in a complex network environment. To overcome this limitation, Shanmugam et al. [23] proposed a content caching mechanism for each node to minimize the expected download latency with general content prevalence and network topology. Balasubramanian et al. [24] focuses on MEC environments for Device-to-Device (D2D) communication and proposes a dynamic caching allocation method to jointly minimize both caching costs and service latency. Recently, many scholars used reinforcement learning such as Deep Q-network (DQN) and Deep Deterministic Policy Gradient (DDPG) to solve critical problems in edge caching systems [25]. To address the long-term popularity and short-term relevance of user requests, Qian et al. [11] proposed a hierarchical reinforcement learning method based on Q-learning and DQN. This method can accurately push or cache appropriate content in MEC networks. He et al. [26] proposed a DRL method that dynamically coordinates network, cache, and computational resources to improve the overall performance of vehicular networks. Li et al. [12] developed a DDPG-based content caching algorithm to optimize the content caching policy, minimizing the average transmission delay and energy consumption of user-requested content in MEC networks. Wu et al. [13] modeled the cache update problem as a Markov decision process and proposed an efficient DRL algorithm based on long short term memory (LSTM) networks and external memory to enhance the caching decision capability of edge servers. The algorithm improves the cache hit ratio and long-term system payoff significantly. Nevertheless, none of the above approaches consider collaborative caching among edge servers. When the base stations in the edge network make caching decisions independently, they may cache the hotspot contents repeatedly, resulting in underutilization of caching resources [14].

Cooperative edge caching has emerged as a prominent research topic in edge caching in recent years. In a multi-BS collaborative caching scenario, content can be shared among edge servers and user requests can be served by the associated edge servers or their neighboring edge servers [15]. Garg et al. [27] designed the MARL algorithm to investigate edge caching with different levels and evaluated it under full cooperation, episodic cooperation, distributed cooperation, and independent operation scenarios. Wu et al. [28] proposed a multi-agent variant of the soft actor-critic-based framework for caching updates in IoT sensing networks. The performance of their proposed distributed cooperative caching scheme is significantly better than that of the centralized caching scheme. Considering the delay and energy balance in MEC networks, Li et al. [29] introduced a DQN-based cooperative edge caching approach that employs a Multilayer Perceptron network to predict video content requested by mobile users and obtains the optimal edge caching policy by a branch-and-bound method. Zhang et al. [30] presented a multi-agent Actor-Critic-based algorithm that considers the application scenario of the varying size of cached data items. Radenkovic et al. [31] proposed a cooperative edge caching method based on the MADRL CognitiveCache and validated it in two real-world network topologies with significant differences in user preferences in New York and London. Their results confirm that the proposed method can adapt to the changing MEC network. Chen et al. [32] addressed the collaborative caching problem as a multi-agent decision problem by using a variational recurrent neural network to predict time-varying user requests in each BS and an LSTM network to predict user movement and location.

The above research primarily focuses on designing cooperative edge caching strategies based on the popularity of predictable or known static content. In practical scenarios, however, content popularity distribution may change dynamically over time, and the assumption of static content popularity is too idealistic [33]. While some work addresses dynamic characteristics of content popularity [34], their approach still assumes a specific Zipf distribution and requires retraining for new content popularity distribution. There is little research on improving caching performance by considering inter-collaboration among edge servers. Furthermore, most studies rely on specific user request popularity distributions and lack generalization capabilities when facing new popularity distributions.

Table 1
Summary of main notations.

Notation	Description
$\mathcal{B}$, B	the set, number of BSs
K_b	caching capacity of BS b
W	Channel bandwidth of BSs
$\mathcal{N}$, N	the set, number of users
W_n	bandwidth allocated to user n
$\mathcal{N}_b$	set of users covered by BS b
$r_{b,n}$	downlink transmission rate
$\mathcal{S}$, S	the set, number of contents
m_s	the size of content s
p_s	probability of user request content
$t \in T$	time slot
$C_b(t)$	content cache state of BS b at slot t
$h^n_{b,s_n}(t)$	cache state of request s_n at slot t
$d^n_b(t)$	delay from BS b to user n at slot t
$S_b(t)$, $A_b(t)$, $R_b(t)$	state, action, reward of BS b
$\mathcal{S}(t)$, $\mathcal{A}(t)$, $\mathcal{R}(t)$	global state, action, reward
e	energy consumption per data bit
$e_b(t)$	energy consumption of BS b at slot t

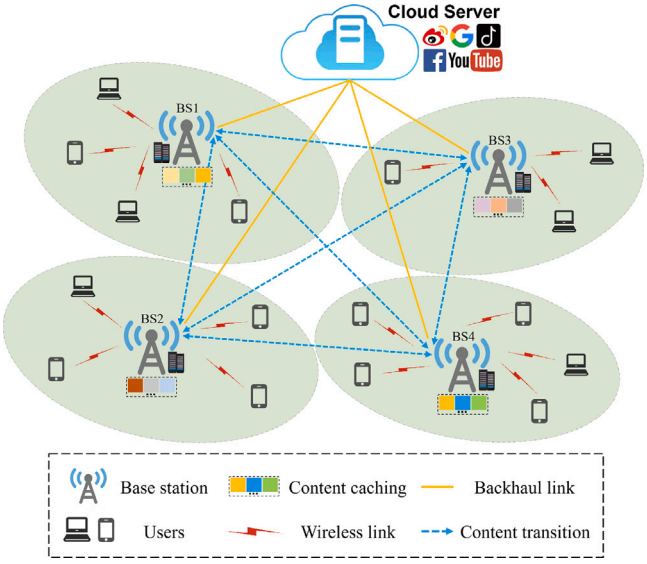


Fig. 1. The cooperative edge caching network.

3. System model and problem formulation

3.1. System model

Table 1 lists the main variables used in our description. Our MEC network consists of a cloud server and B BSs that cater N users. The cloud server has ample storage to cache all available services. The BS has limited caching capabilities, which allows it to cache only a few services. The BSs work collaboratively to fulfill content requests from users. The network structure is shown in Fig. 1. The user request is uploaded to a BS, which identifies a content-fetching location based on the requested content and its cache status.

Time is divided into T slots, where t is a time slot. Each BS updates its cached content at the beginning of each time slot. BS b, $b \in \mathcal{B}$ and $\mathcal{B} = \{b_1, b_2, \dots b_B\}$, is configured with one server, and its maximum caching capacity is denoted as K_b. The coverage area of BSs has overlaps, and users n, $n \in \mathcal{N}$ and $\mathcal{N} = \{n_1, n_2, \dots, n_N\}$, in the overlapping area select the BS with the best channel conditions for the association. The set of users associated with BS b is denoted as $\mathcal{N}_b$, and all the users are concatenated as $\mathcal{N} = \cup_{b\in\mathcal{B}}\mathcal{N}_b$. MEC scenarios can provide S types of content for user requests, and we denote a specific content as s, $s \in \mathcal{S}$ and $\mathcal{S} = \{s_1, s_2, \dots, s_S\}$. The size of content s is

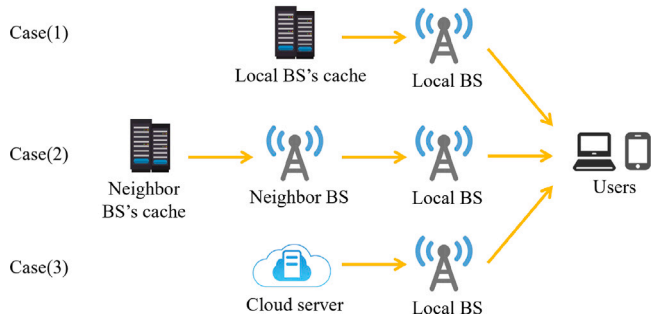


Fig. 2. The transmission paths for the different requested content.

$m_s \in [m_{\min}, m_{\max}]$, where m_{max} and m_{min} are the upper and lower bounds, respectively.

3.1.1. Caching model

The content caching state of all BSs at time slot t is denoted by $C(t) = \{C_b(t) \mid b \in \mathcal{B}\}$, where $C_b(t) = \{c_{b,s}(t) \mid s \in \mathcal{S}\}$ represents the content caching state of BS b. The binary variable $c_{b,s}(t)$ indicates whether BS b caches content s at slot t; $c_{b,s}(t) = 1$ means the content is cached, while $c_{b,s}(t) = 0$ means contrast. Each BS must abide by the cache capacity limit when performing content caching

$$\sum_{s\in\mathcal{S}} c_{b,s}(t) m_s \le K_b, c_{b,s}(t) \in \{0,1\}, \forall b \in \mathcal{B}. \tag{1}$$

Adding cached content to a BS requires downloading from the cloud server, resulting in energy consumption. In the MEC network, we only consider the energy consumption of adding new contents and ignore the energy consumption required to overwrite or delete previously cached content.

3.1.2. Request model

In the system, the Zipf distribution is commonly used to model the popularity of user requests for content [1]. The content requests of all users are independent and obey the same distribution. We can represent the probability of user requests for all content using the vector $P = \{p_s \mid s \in \mathcal{S}\}$. p_s denotes the probability of a user requesting s and can be calculated by

$$p_s = \frac{{u_s}^{-\theta}}{\sum_{s\in\mathcal{S}} {u_s}^{-\theta}}, \tag{2}$$

where u_s is the ranking of s in terms of frequency of being requested among all contents, and θ is the skewness factor. A higher value of θ indicates more concentrated content popularity.

As shown in Fig. 2, a user obtains the requested contents directly if the request has been cached in local BS. However, if the contents are not cached locally, the neighbor BSs can be utilized to retrieve the contents. Otherwise, the contents are fetched from the cloud server.

3.1.3. Communication model

BSs utilize the Orthogonal Frequency Division Multiple Access techniques to facilitate communication with users. Each associated user is allocated a dedicated channel for data transmission by the BS. Furthermore, the utilization of this technique ensures that there are no interference issues between different transmission channels [35]. The downlink transmission rate $r_{b,n}$ between user n and BS b is calculated according to Shannon's formula [36]:

$$r_{b,n} = W_n lb(1 + \frac{P_n h_n}{\sigma^2}), \tag{3}$$

where $W_n = W/|\mathcal{N}_b|$ denotes the transmission bandwidth allocated by BS b to user n, W is the channel bandwidth of BS, and $|\mathcal{N}_b|$ denotes the number of users associated with BS b. The channel gain of user n

is denoted by h_n, while P_n represents the transmission power of user n, and σ^2 denotes the noise power. The BSs are connected through a Local Area Network [35], and the transmission rate between two BSs is denoted as a fixed value $r_{b,b}$. The transmission rate between a BS and the cloud server is also fixed and denoted as $r_{c,b}$.

When user n sends a content request s_n to BS b at time slot t, its cache hit status can be expressed as

$$h^n_{b,s_n}(t) \in (h_0, h_1, h_2), \tag{4}$$

where $h_0 = \begin{bmatrix}1 & 0 & 0\end{bmatrix}^T$, $h_1 = \begin{bmatrix}1 & 1 & 0\end{bmatrix}^T$ and $h_2 = \begin{bmatrix}1 & 0 & 1\end{bmatrix}^T$ denote cache hits for s_n occurring at the local BS, neighbor BS, and cloud server, respectively. In the MEC network, the transmission delay for the content request signal is significantly smaller than the content transmission delay, thus it can be disregarded. Consequently, the delay for user n to receive the requested content s_n from BS b at time slot t can be computed as follows:

$$d^n_b(t) = \begin{bmatrix} \frac{m_{s_n}}{r_{b,n}} & \frac{m_{s_n}}{r_{b,b}} & \frac{m_{s_n}}{r_{c,b}} \end{bmatrix} \cdot h^n_{b,s_n}(t). \tag{5}$$

3.2. Problem formulation

To minimize the content transmission delay and the energy consumption caused by cache replacement, we introduced the system utility Q as a weighted sum of the delay and energy consumption:

$$Q = \sum_{t\in T}\sum_{b\in\mathcal{B}}\left[\sum_{n\in\mathcal{N}_b} d^n_b(t) + \delta e_b(t)\right], \tag{6}$$

where a represents the current task, $e_b(t)$ denotes the energy consumption resulting from the cache replacement action performed by BS b during time slot t, and δ is the weight associated with delay and energy consumption.

This study aims to explore the generalization performance of cooperative edge caching policies to minimize the long-term system utility:

$$\begin{aligned} &\min \sum_{f\in\mathcal{F}} Q_f \\ &s.t.\,(1)\ and\ (4) \end{aligned}, \tag{7}$$

where $\mathcal{F}$ denotes the set of test tasks used to assess the caching policy's generalization capability when confronted with new tasks. The popularity distribution of the content in the test task is novel and has not been observed during the algorithm's training on previous training tasks.

4. Multi-agent deep reinforcement learning model for cooperative caching

The state of the MEC network in the next time is determined by the current state and caching decision. Therefore, a Markov process is appropriate for modeling the state transfer. In a distributed caching policy, BS can only access local observation information and has no access to the current user request state because caching decisions are made before user requests arrive. As a result, the optimization problem in Eq. (7) is a multi-agent decision problem based on the partially observable Markov decision process (POMDP) [37], where each BS is considered an agent. A POMDP can be represented by the tuple $(\mathcal{B}, \mathcal{S}, \mathcal{A}, T, R, \mathcal{O}, \gamma)$, where $\mathcal{B}$ is the set of agents, $\mathcal{S}$ and $\mathcal{A}$ are the sets of states and actions of all agents, respectively. T represents the conditional transition probability between states, R is the reward function, $\mathcal{O}$ denotes the set of local observations of all agents, and $\gamma \in [0, 1]$ is the reward discount factor.

4.1. Observation and state space

The local state of the agent b at time slot t is

$$S_b(t) = \left\{C_b(t), M_b(t), X_b(t)\right\}, \tag{8}$$

which consists of a local cache state C_b, content size M_b, and user request state X_b. It is noteworthy that $M_b(t) = \{m_{b,1}(t), m_{b,2}(t), \ldots, m_{b,S}(t)\}$ only records the size of cached contents in BS b and the size of other content is recorded as zero.

When an agent performs a cache replacement, it cannot get the user request status of the current time slot, but only the user's request history. The number of all content history requests observed by agent b at time slot t is $\hat{X}_b(t) = \{\hat{x}_{b,1}(t), \hat{x}_{b,2}(t), \ldots, \hat{x}_{b,S}(t)\}$. For simplicity, we transfer $\hat{X}_b(t)$ to the range of [0, 1] by linear normalization. The local observation of agent b at time slot t is

$$O_b(t) = \left\{C_b(t), M_b(t), \hat{X}_b(t)\right\}. \tag{9}$$

The global state for the whole system at time slot t is defined as $\mathcal{S}(t) = \{S_1(t), \ldots, S_b(t), \ldots, S_B(t)\}$, where $S_b(t) = O_b(t)$.

4.2. Action space

At the beginning of each time slot, the agents need to update the local cache. The cache replacement action $A_b(t)$ by agent b at time t is

$$A_b(t) = \{a_{b,1}(t), \ldots, a_{b,s}(t), \ldots, a_{b,S}(t)\}, a_{b,s} \in \{0, 1\}, \tag{10}$$

where $a_{b,s} = 1$ is that agent b decides to cache content s, $a_{b,s} = 0$ represents not caching. Action $A_b(t)$ satisfies the constraint Eq. (1), and the global action for the system at time slot t is defined as $\mathcal{A}(t) = \{A_1(t), A_2(t), \ldots, A_B(t)\}$.

4.3. Reward space

Frequent cache replacement leads to a considerable increase in network communication overhead. It is crucial to minimize the energy consumption associated with updating BS cached content while optimizing the latency for users to access content. The energy consumption of performing action A_b for agent b is

$$e_b(t) = e \cdot \frac{1}{2}\sum_{s\in\mathcal{S}}(a_{b,s}(t) - c_{b,s}(t))(a_{b,s}(t) - c_{b,s}(t) + 1) \cdot m_s, \tag{11}$$

where e is the energy consumption per unit size of cached content, $c_{b,s}(t)$ is the cache state of content s in BS b when the execution action is performed.

At time slot t, the immediate reward that agent b receives for executing a cache replacement action is determined by a weighted combination of delay reduction and energy conservation

$$R_b(t) = \sum_{n\in\mathcal{N}_b}\begin{bmatrix} \frac{m_{s_n}}{r_{b,n}} & \frac{m_{s_n}}{r_{b,b}} & \frac{m_{s_n}}{r_{c,b}} \end{bmatrix} \cdot h_2 - \sum_{n\in\mathcal{N}_b} d^n_b(t) - \delta \cdot e_b(t), \tag{12}$$

where delay reduction refers to the reduced delay compared to obtaining all the content from the cloud server. The global reward for the system at time slot t is defined as $\mathcal{R}(t) = \{R_1(t), \ldots, R_b(t), \ldots, R_B(t)\}$. And the overall reward $R(t)$ for the entire MEC network can be immediately expressed as

$$R(t) = \sum_{b\in\mathcal{B}} R_b(t). \tag{13}$$

In our multi-agent system, the state value function $V_\pi(s)$ that considers long-term rewards is the cumulative expected reward under the joint policy π:

$$V_\pi(s) = \mathbb{E}\left[\sum_{t\in T}\gamma^t R(t)\middle|S(t), \pi\right], \tag{14}$$

where $\pi = \{\pi_1, \pi_2, \ldots, \pi_B\}$ is the set of caching policies for all agents, and $\gamma \in [0, 1]$ is the discount factor. The value of γ influences the

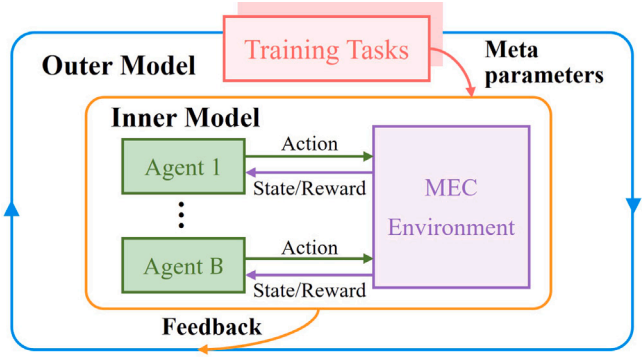


Fig. 3. The framework of MRL-based MAMRC.

extent to which future rewards affect present decisions. Higher values of γ prioritize long-term rewards. Hence, the problem of optimizing cooperative edge caching policy is finding the policy π^* to maximize long-term reward of the system.

5. Meta reinforcement learning-based cooperative edge caching algorithm

Fig. 3 illustrates the basic framework of MAMRC, which consists of two main models: the inner model and the outer model. During MAMRC learning, the inner model is initialized with meta parameters provided by the outer model. It is then trained using reinforcement learning to acquire the optimal cache replacement policy tailored to the current sampling task. The learned model parameters of the inner model are fed back to the outer model, which adjusts the meta-parameters based on this feedback. As a result, the inner model is initialized with updated meta-parameters and can be promptly adapt to new tasks.

5.1. Inner model

The framework of the inner model is shown in Fig. 4. The inner model extends the MADDPG algorithm [38] and facilitates cooperative caching among BSs through centralized training and distributed execution. When the set of global states $\mathcal{S}$ of the MEC network is observed, each agent makes cache content replacement decisions based on their local observations. The MEC network provides immediate cache rewards $\mathcal{R}$ after all agents have executed their actions $\mathcal{A}$. After cache replacement, the MEC network state is updated to $\mathcal{S}'$, and the state transfer tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{S}')$ is stored in the replay buffer D.

Each agent comprises multiple deep neural networks, including the actor network $\pi(\theta)$ and critic networks $Q^{\pi}(\phi)$, along with their corresponding target networks $\hat{\pi}(\hat{\theta})$ and $\hat{Q}^{\hat{\pi}}(\hat{\phi})$. The target networks are introduced to maintain stability when inner model training and network structure are identical to the original network.

At time slot t, the actor network parameters of agent b are defined as $\theta_b(t)$. π_b is the cache replacement policy function that maps the local state $S_b(t)$ of the agent to the cache replacement action $A_b(t)$

$$A_b(t) = \pi_b(S_b(t)\big|\theta_b(t)). \tag{15}$$

To facilitate the exploration of potentially optimal cooperative caching strategies, the actor introduces the Gumbel-SoftMax Trick mechanism by adding random noise. Furthermore, the final output vector of the actor network is denoted as $\hat{A}$, which has to be mapped to an executable cache replacement action A using Algorithm 1.

Algorithm 1 Caching action mapping

Require: The outpour vector of the actor network $\hat{A}_b, \forall b \in \mathcal{B}$;
Ensure: The adjusted caching action for agent b: A_b;
1: Initialize $A_b = \{a_{b,1}, \cdots, a_{b,s}, \cdots, a_{b,S}\}$, where $a_{b,s} = 0, \forall s \in \mathcal{S}$, $\Delta = 0$, $i = 1$;
2: **while** $\Delta < K$ **do**
3: Get the ith largest index s of vector $\hat{A}_b$;
4: **if** $\Delta + m_s < K$ **then**
5: $a_{b,s} = 1$, $i = i + 1$;
6: **end if**
7: $\Delta = \Delta + m_s$;
8: **end while**

The critic network serves as an evaluator of the cache replacement actions. At time t, the critic network receives the locally observed states $\mathcal{S}(t)$ and the set of cache replacement actions $\mathcal{A}(t)$ performed by all agents. The state value Q_b^{π} is computed as follows:

$$Q_b(t) = Q_b^{\pi}(\mathcal{S}(t), \mathcal{A}(t)\big|\phi_b(t)), \tag{16}$$

where $\phi_b(t)$ denotes the critic network parameter of agent b.

At regular intervals, $\{(\mathcal{S}^j, \mathcal{A}^j, \mathcal{R}^j, \mathcal{S}'^j)\big|j \in \mathcal{H}\}$ are randomly sampled from the reply buffer and input into the network for training, where $\mathcal{H}$ is the set of sample indices. The actor updates the network parameters using policy gradient, while the critic network parameters are updated by computing the temporal difference (TD) error.

Taking agent b as an example, the parameters of actor network θ_b is updated by

$$\theta_b \leftarrow \theta_b + \alpha \cdot \nabla_{\theta_b} J(\theta_b), \tag{17}$$

where α is the learning rate of the actor network, and $\nabla_{\theta_b} J(\theta_b)$ refers to the policy gradient. When the batch size is H, $\nabla_{\theta_b} J(\theta_b)$ is calculated as

$$\begin{aligned}\nabla_{\theta_b} J(\pi_b) &= \mathbb{E}_{S,A\sim D}[\nabla_{\theta_b}\pi_b(A_b\big|S_b)\nabla_A Q_b^{\pi}(S, A_1, \ldots, A_b, \ldots, A_B)\big|_{A_b=\pi_b(s_b\big|\theta_b)}] \\ &\approx \frac{1}{H}\sum_{j\in\mathcal{H}} \nabla_{\theta_b}\pi_b(S_b^j\big|\theta_b)\nabla_{A_b^j} Q_b^{\pi}(S^j, A_1^j, \ldots, A_b^j, \ldots, A_B^j)\big|_{A_b^j=\pi_b(s_b^j\big|\theta_b)}.\end{aligned} \tag{18}$$

The critic network is trained using TD error by minimizing the loss function

$$\mathcal{L} = \frac{1}{H}\sum_{j\in\mathcal{H}} (y_b^j - Q_b^{\pi}(S^j, A_1^j, \ldots, A_b^j, \ldots, A_B^j))^2. \tag{19}$$

When the reward discount factor is γ, y_b^j is calculated as

$$y_b^j = R_b^j + \gamma \cdot \hat{Q}_b^{\hat{\pi}}(S'^j, A_1'^j, \ldots, A_b'^j, \ldots, A_B'^j\big|\hat{\phi})\big|_{A_b'^j=\hat{\pi}_b(s_b'^j\big|\hat{\theta}_b)}. \tag{20}$$

The target network is not involved in the training process. After the actor and critic networks update their parameters, the parameters of the target network are soft updated as

$$\begin{aligned}\hat{\theta}_b &\leftarrow \tau\cdot\theta_b + (1-\tau)\cdot\hat{\theta}_b \\ \hat{\phi}_b &\leftarrow \tau\cdot\phi_b + (1-\tau)\cdot\hat{\phi}_b\end{aligned}, \tag{21}$$

where τ is the smoothing coefficient. The complete process of the proposed cooperative edge caching algorithm is shown in Algorithm 2.

5.2. Outer model

By combining the objective function of Model-Agnostic Meta-Learning (MAML) [39] with the training method of the inner model, the objective function of the outer model is

$$\mathcal{L}^{Outer}(\psi) = \mathbb{E}_{\mathcal{T}_i\in\rho(\mathcal{T})}\left[\mathcal{L}_{\mathcal{T}_i}^{Inner}(\xi_i')\right], \xi_i' = f\left(\xi_i, \mathcal{T}_i\right), \xi_i = \psi \quad , \tag{22}$$

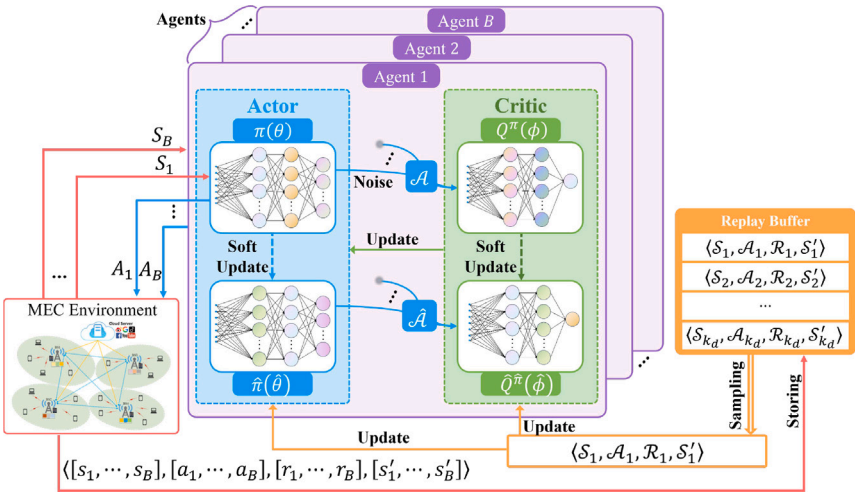


Fig. 4. The framework of MADRL-based inner model.

Algorithm 2 MARL-based inner model

Require: The actor network π_b and target network $\hat{\pi}_b$ with random parameters θ_b; the critic network Q_b^{π} and target network $\hat{Q}_b^{\hat{\pi}}$ with random parameters ϕ_b; the reply buffer D with memory capacity K_d;
Ensure: optimal caching policy π^*;
Random initialize the content cache policy of all BSs;
for slot=1 to T **do**
  initialize t=1;
  for all agent $b \in \mathcal{B}$ **do**
    Receive the local observation $S_b(t)$;
    Get the outpour vector of actor $\hat{A}_b(t) = \pi_b(S_b(t)|\theta_b(t))$;
    Obtain adjusted action $A_b(t)$ using Algorithm 1;
    Execute action $A_b(t)$ and update the cache policy of BS b;
    Receive the user requests this slot;
    Calculate the immediate reward $R_b(t)$ using (12);
    Receive the subsequent observation $S_b'(t)$;
  end for
  Store $(S(t), \mathcal{A}(t), \mathcal{R}(t), S'(t))$ in D;
  if t%I=0 **then**
    Sample random H samples $(S^j, \mathcal{A}^j, \mathcal{R}^j, S'^j), j \in \mathcal{H}$ from D;
    for all agent $b \in \mathcal{B}$ **do**
      Update actor π_b using the stochastic gradient descent (18) by Adam;
      Update critic Q_b^{π} by minimizing the loss (19) by Adam;
      The parameters of target network $\hat{\pi}_b$ and $\hat{Q}_b^{\hat{\pi}}$ are soft updated by (21);
    end for
  end if
  $t = t + 1$;
end for

where $f(\xi_i, \mathcal{T}_i)$ represents the inner model update function that is initialized with the meta parameter ψ. $\mathcal{T}_i$ denotes the ith task sampled in the outer model training phase, while ξ_i represents the inner model initialization parameter corresponding to $\mathcal{T}_i$. ξ_i' refers to the network parameter when $\mathcal{T}_i$ is trained to obtain the optimal caching policy.

The stochastic gradient descent is used to optimize the objective function and the meta parameters ψ are updated as follows:

$$\psi \leftarrow \psi - \lambda \nabla_{\psi} \sum_{\mathcal{T}_i \in \rho(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}^{Inner}(\xi_i'), \tag{23}$$

where λ is the learning rate of outer model training.

However, optimizing the objective function using Eq. (22) requires the calculation of second-order gradients, which is time-consuming and computationally expensive. In addition, owing to the actor and critic networks in the inner model using different methods to update network parameters, it is challenging to compute second-order gradients. To address these issues, the outer model optimizes Eq. (22).

Fig. 5 illustrates the workflow of the outer model. In the training phase, after each task has learned a content caching strategy, the network parameters are updated using the difference between the parameters before and after the task:

$$\begin{cases} \psi_{actor} \leftarrow \psi_{actor} + \lambda \frac{1}{n} \sum_{i=1}^{n} (\xi'_{i,actor} - \psi_{actor})/\alpha \\ \psi_{critic} \leftarrow \psi_{critic} + \lambda \frac{1}{n} \sum_{i=1}^{n} (\xi'_{i,critic} - \psi_{critic})/\beta \\ \psi_{target\_actor} \leftarrow \tau \cdot \psi_{actor} + (1-\tau) \cdot \psi_{target\_actor} \\ \psi_{target\_critic} \leftarrow \tau \cdot \psi_{critic} + (1-\tau) \cdot \psi_{target\_critic} \end{cases}, \tag{24}$$

where n is the batch size in the outer model training, α and β are the learning rates of the actor network and the critic network of the inner model, respectively. The smoothing coefficient τ is used to update target networks. The process of updating meta-parameters is shown in Fig. 6, where the arrows depict the direction of gradient updates.

Algorithm 3 presents the overall design of MAMRC. Initially, a batch of tasks is sampled from the training task set $\rho(\mathcal{T})$, and each task is trained using Algorithm 3 to learn the optimal cache replacement policy. The meta parameters are updated using Adam [40]. Once the outer model is trained to obtain meta parameters with stable effects, the inner model does not require further training to adapt to tasks and make caching decisions.

Algorithm 3 MRL-based MAMRC

Require: Task distribution $\rho(\mathcal{T}) = \{\mathcal{T}_1; \cdots, \mathcal{T}_i, \cdots, \mathcal{T}_I\}$;
Ensure: Meta parameter ψ;
Random initialize meta parameter ψ;
for iterations $k \in \{1, \cdots, K\}$ **do**
  Sample n tasks $\mathcal{T}_i \in \{\mathcal{T}_1, \mathcal{T}_2, \cdots, \mathcal{T}_n\}$ from $\rho(\mathcal{T})$;
  for each task $\mathcal{T}_i \in \{\mathcal{T}_1, \mathcal{T}_2, \cdots, \mathcal{T}_n\}$ **do**
    Initialize the parameters of $\mathcal{T}_i$: $\xi_i \leftarrow \psi$;
    Train $\mathcal{T}_i$ using Algorithm 2;
  end for
  Update ψ using (24) via Adam;
end for

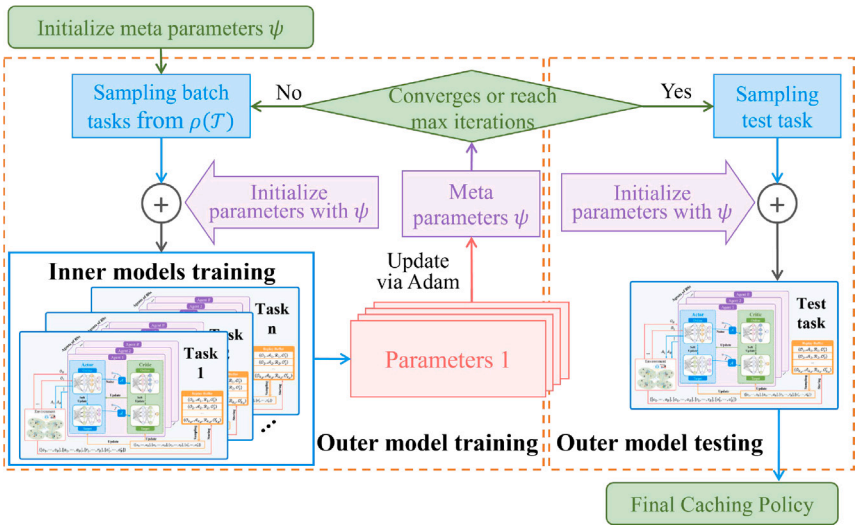


Fig. 5. The workflow of the outer model.

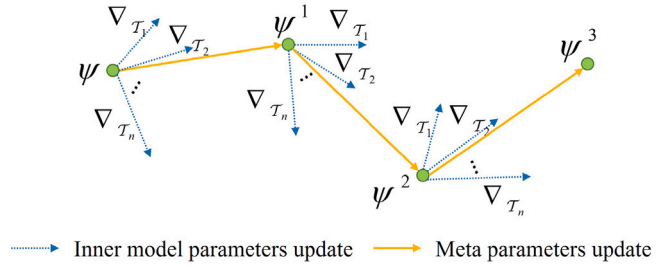


Fig. 6. The update process of meta parameters.

6. Performance evaluation

6.1. Simulation setup

In the MEC scenario, services are provided to users by four BSs that communicate with each other. The bandwidth for each BS is 20MHz, which is divided equally among its users. Each BS has an edge server with a content cache capacity of 200 MB. The number of users associated with the four BSs is 50, 30, 20, and 40. To replicate real-world user requests, the experiments are conducted using the MovieLens 1M Dataset [41]. Following the methodology in [34], each content's frequency of request is approximated using a Zipf distribution with a skewness factor of 0.8. In each time slot, the users send requests to a BS with a total of S=1000 request types in the network, and each content has a size $m \in [1, 10]$ MB. Additional simulation parameters are detailed in Table 2.

The experiment was based on the deep learning framework PyTorch 1.12.0, written in Python 3.7.0. The computer hardware configuration comprised an AMD Ryzen 7 5800H CPU, NVIDIA GeForce RTX 3060 GPU, and 32 GB memory, with Windows 10 operating system. We modeled each BS as an agent in the experiments, incorporating the actor network, critic network, and their corresponding target networks. Each deep neural network contains two fully connected layers, with the first layer consisting of 128 neurons and the second of 256 neurons, using the ReLU activation function. The learning rates for the actor and critic networks are 0.0001 and 0.0006, respectively. The Adam optimizer was used for updating, with network parameters updated every hundred-time slots. The detailed hyperparameters settings for the MAMRC training are shown in Table 2.

Table 2
Evaluation parameters.

Parameter	Value
Number of BSs	4
Number of users	50, 30, 20, 40
Number of contents	1000
Content size	[1,10] MB
Skewness factor	0.8
Bandwidth of BSs	20 MHz
Transmission rate between BSs	5 MB/S
Transmission rate between the cloud server and BS	1 MB/S
Transmission power between user and BS	0.1 W
Gauss white noise power	100 dBm
Unit energy consumption for content transmission	0.3 J/MB
Weight of energy consumption and delay	0.5
Number of steps in each episode	100
Inner and outer model batch size	128, 8
Memory capacity of reply buffer	10^5
Actor and critic learning rate	0.0001, 0.0006
Reward discount factor	0.96
Target smoothing coefficient	0.01
Outer model learning rate	2×10^{-6}

6.2. Verification of inner model

We conducted a comparative analysis with four popular edge caching methods, LFU, FIFO, DDPG, and MAAC.

- **LFU**: Select the content with the lowest frequency of user requests within a specific period for replacement.
- **FIFO**: Selects the earliest cached content for replacement.
- **DDPG** [12]: A deterministic policy gradient algorithm based on actor-critic, where the agent independently formulates the cache replacement strategy.
- **MAAC** [30]: A multi-agent actor-critic caching algorithm inspired by the Actor-Critic algorithm, which is designed to tackle the cooperative caching issue.

LFU and FIFO are traditional cache replacement methods that update caching based on the content's access frequency and arrival time. In contrast, DDPG is a DRL method that utilizes the same state, action, and reward function settings as the inner model. However, it functions without considering the collaboration between BSs. MAAC is an edge caching algorithm based on MADRL. Apart from sharing the same reward function as the inner model, all other algorithmic settings are

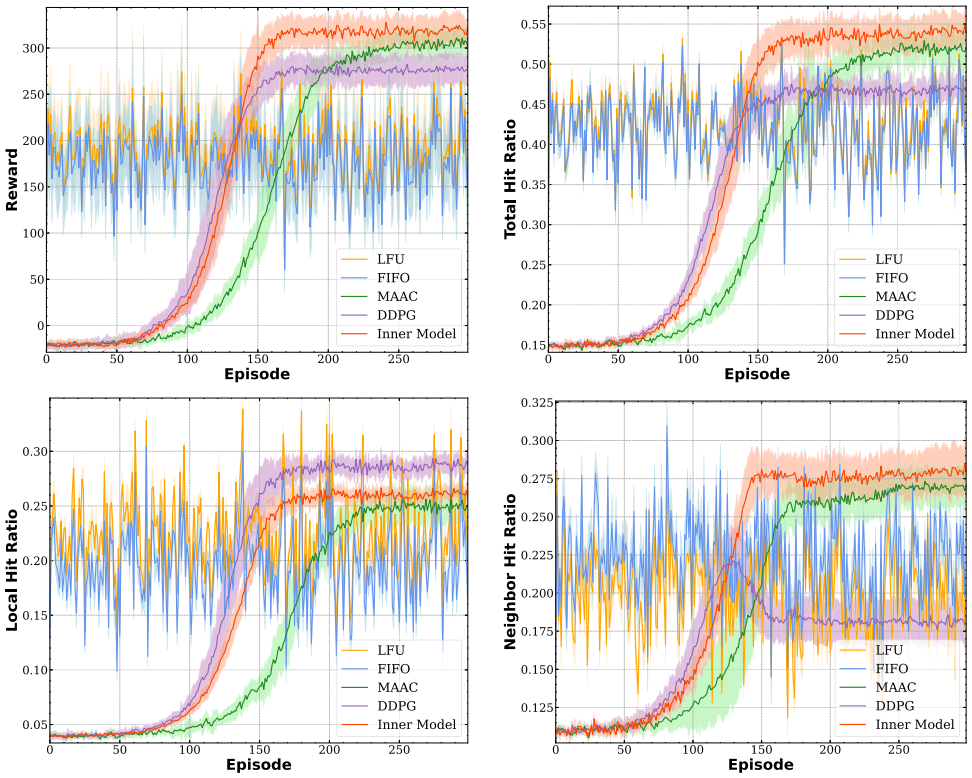


Fig. 7. Performance comparison of different caching methods.

Table 3
Performance of different methods.

Method	Reward	Total hit ratio	Local hit ratio	Neighbor hit ratio
LFU	190.90	0.4244	0.2257	0.1987
FIFO	177.86	0.4186	0.1966	0.2220
DDPG	275.85	0.4671	**0.2860**	0.1811
MAAC	304.33	0.5192	0.2498	0.2693
MAMRC	**318.19**	**0.5378**	0.2602	**0.2776**

referenced from [30]. We use the same sequence of user requests in our experiments to ensure a fair comparison.

In Fig. 7, the horizontal axis represents the episode of network parameter updates, each consisting of one hundred time slots. The vertical axis shows the reward function values and the total, local, and neighbor cache hit rates. Fig. 7(a) and Fig. 7(b) show that the performance of cache replacement methods, LFU and FIFO, is far inferior to DRL methods. Both DDPG and the inner model converge at around 170 episodes, while MAAC requires training for approximately 250 episodes to achieve stable results. DDPG exhibits a significantly lower average reward and total hit ratio than the MADRL methods that consider content-cooperative caching. In comparison to MAAC, the proposed inner model converges faster and demonstrates superior performance. Fig. 7(c) and 7(d) show that DDPG has a higher hit ratio for local caches. In contrast, the hit ratio for neighbor caches is relatively lower. This is because, in DDPG, the agent independently formulates cache replacement strategies, which may result in repeatedly caching popular content. Despite experiencing a reduction in the local hit ratio, the performance of the inner model considering content collaboration caching is significantly better than other baseline methods.

The results in Table 3 are the average of multiple time slots after performance stabilization. The immediate reward of the inner model is improved by 66.68%, 78.90%, 15.35% and 4.55% compared with LFU, FIFO, DDPG and MAAC, respectively, and the total cache hit ratio is improved by 26.72%, 28.48%, 15.14%, and 3.58% respectively. As the traditional cache replacement mechanism brings massive energy consumption by frequently updating the cache contents, the reward improvement of the inner model is more significant than that of the total hit ratio.

To evaluate the caching performance of the inner model in different MEC networks, we conducted experiments with varying numbers of BSs, total content numbers, BS cache capacities, and Zipf distributions. Other network parameters follow the values in Table 2, and the results are illustrated in Fig. 8. As shown in Fig. 8(a), the long-term rewards of all methods increase with the number of BSs, where the reward of the inner model is significantly better than others. Fig. 8(b) displays the performance under different content amounts ranging from 500 to 2500 with a step size of 500. It can be observed that the cache performance becomes worse as the content amount increases. Larger cache space allows BSs to cache more content. Hence, the curve in Fig. 8(c) shows an upward trend but is not linear. In Fig. 8(d), we varied the Zipf skewness factor between 0.2 and 1.0 with a step of 0.2 and observed that the long-term rewards of all methods exhibited an upward trend. This finding suggests that an increase in the concentration of user requests for popular content leads to an improvement in cache hit rates. In summary, the inner model's cache performance performs optimally in various MEC environments.

6.3. Verification of MAMRC

In the training process of MAMRC, the content popularity of the training and testing tasks follows different distributions, as shown in Table 4. Specifically, the popularity distribution in the testing task was distinct from that of the training task, aiming to evaluate the generalization capability of MAMRC when dealing with new tasks. To better mimic the real-world scenarios, we randomly selected the number of users and the size of contents in training and test tasks within the ranges provided in Table 4.

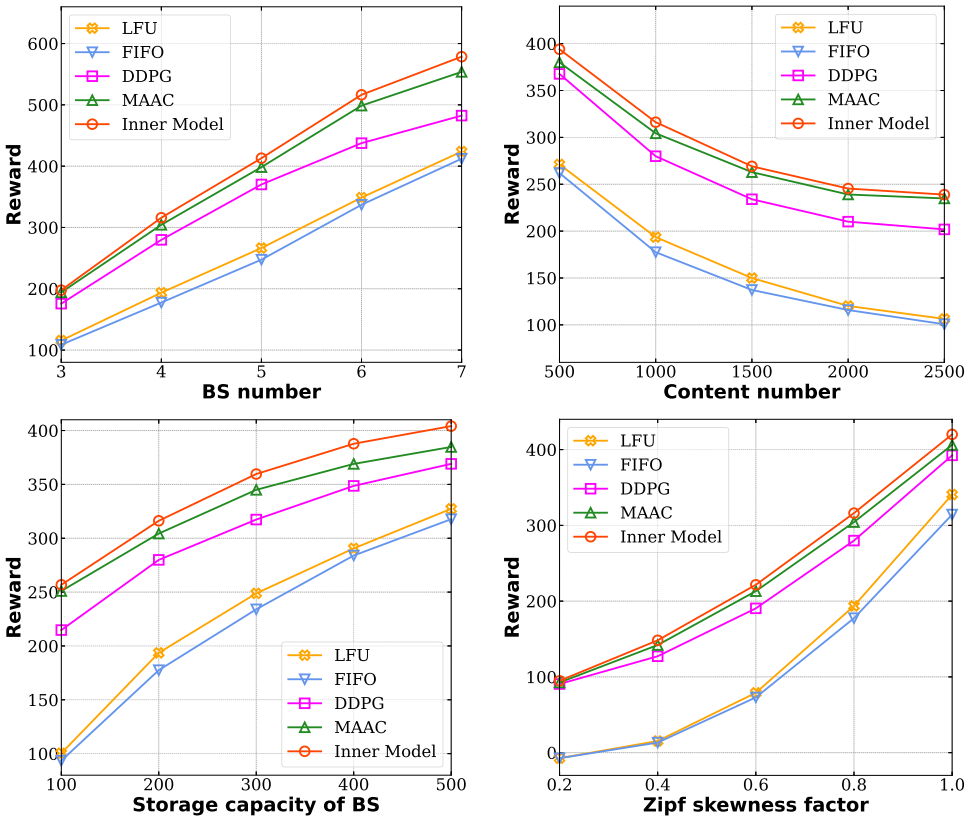


Fig. 8. Performance comparison of different MEC environments.

Table 4
Evaluation parameters setup of outer model.

Parameter	Training task	Testing task
θ	0.4 – 0.7, 0.8 – 1.0	0.7 – 0.8
N_b	[20, 60]	[20, 60]
m	[1, 10] MB	[1, 10] MB

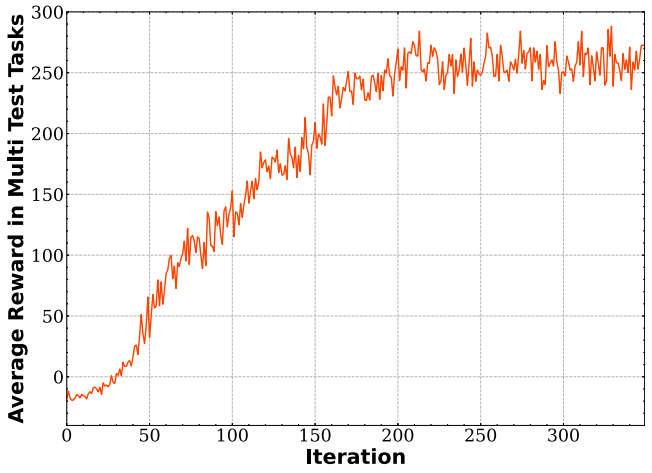


Fig. 9. Convergence of outer model in the training phase.

Hyperparameter tuning experiments have shown that the outer model can achieve the best performance when the training batch size is eight and the learning rate λ is 2×10^{-6}. The training process is depicted in Fig. 9, where the horizontal axis is the iterations of meta-parameter updates, and the vertical axis represents the average value of the long-term reward achieved by the inner model when initialized with the current meta-parameters across various test tasks.

To demonstrate the generalization performance of MAMRC under different tasks, the following models were employed to execute cache replacement policies in the testing tasks.

- **LFU**: Directly using LFU to decide the replacement of cached content.
- **Trained inner model(TIM)**: The inner model that learns the optimal caching policy through a specific training task.
- **MAMRC**: The inner model initialized with meta-parameters at the convergence of the outer model.

Fig. 10 demonstrates the caching performance achieved by the methods above across multiple testing tasks. The horizontal axis represents specific testing tasks, and the vertical axis shows the average reward across multiple time slots in the testing tasks. Due to variations in the number of user requests, the average rewards obtained by the same method may differ across different testing tasks. It can be seen that when facing a new task, the caching performance of the TIM declines severely, and in a few testing tasks, it even falls below that of the LFU algorithm. This is because traditional DRL algorithms have poor generalization abilities, and the model parameters already trained may become invalid when facing completely new tasks. By introducing meta-learning into DRL, MAMRC effectively solves the problem mentioned above, and the caching performance is significantly better than other algorithms. Compared with LFU and TIM, the caching performance in total test tasks is improved by 57.35% and 26.51%, respectively.

For a more intuitive comparison of the generalization performance of models, we selected LFU, multiple different TIMs, and MAMRC as cache decision models and conducted simulations across 20 various testing tasks. The simulation results, as shown in Fig. 11, depict the total hit ratio averaged over multiple time slots on the vertical axis, while the horizontal axis represents different cache decision models. The specific values corresponding to the metrics of each method are

Table 5
Total hit ratio of different methods.

Metric	Methods									
	LFU	TIM1	TIM2	TIM3	TIM4	TIM5	TIM6	TIM7	TIM8	MAMRC
Min	0.3424	0.3831	0.3868	0.3607	0.3514	0.2794	0.2474	0.3482	0.3620	0.4307
Max	0.4289	0.4591	0.4549	0.4305	0.4174	0.3366	0.2925	0.4281	0.4505	0.4699
Fluctuation	0.0865	0.0759	0.0681	0.0697	0.0659	0.0572	0.0451	0.0799	0.0884	**0.0391**
Average	0.3826	0.4184	0.4164	0.3900	0.3754	0.3007	0.2739	0.3882	0.4100	**0.4482**

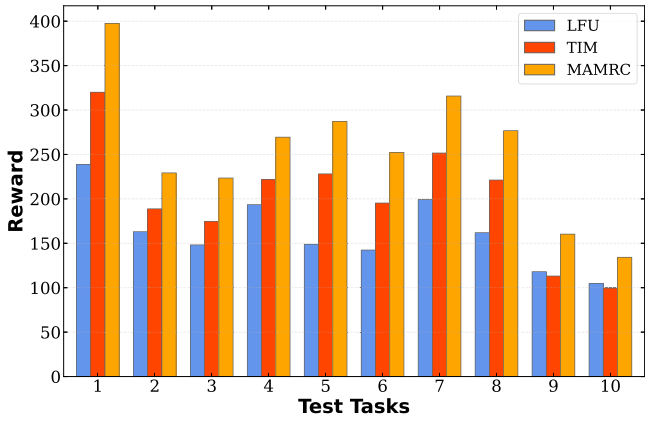


Fig. 10. Performance comparison of different methods in multi test tasks.

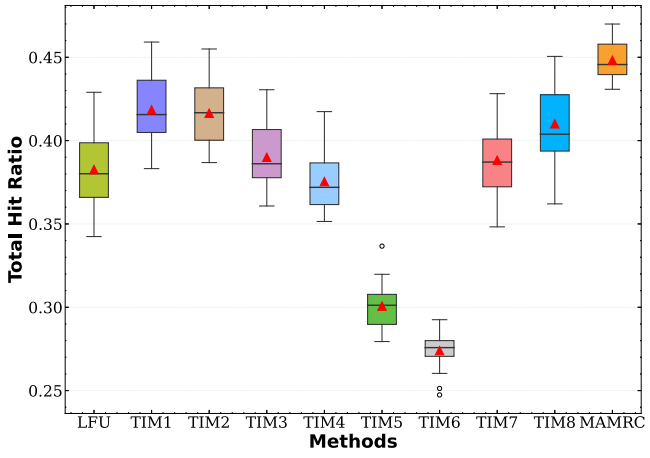


Fig. 11. Generalization performance comparison of different methods.

shown in Table 5. It can be observed that the average total hit ratio of MAMRC is 0.4482. Compared to LFU and multiple TIMs, MAMRC achieves optimal cache performance when facing entirely new testing tasks. Besides, MAMRC exhibits strong generalization capabilities, as it demonstrates the smallest fluctuation range in the overall total hit ratio across different testing tasks, with only 0.0391, while the average fluctuation range for all TIMs is 0.0688.

7. Conclusion and future work

This paper proposes a novel cooperative edge caching algorithm MAMRC based on multi-agent meta-reinforcement learning to overcome the limitations of existing algorithms, such as low performance and poor generalization ability. In this approach, we first model the optimization problem as a POMDP-based multi-agent decision problem and then design a reinforcement learning algorithm to explore the optimal cooperative edge caching policy. Furthermore, we introduce meta-learning to guide DRL using meta-knowledge accumulated from relevant tasks, which effectively enhances the robustness and generalization performance of the DRL algorithm across different task scenarios. Experimental results demonstrate that MAMRC outperforms baseline methods, including greedy and DRL algorithms, achieving optimal caching performance. Moreover, due to the incorporation of meta-parameters, MAMRC demonstrates robust generalization capability when confronted with new tasks.

There is still much room for future research. For example, exploring cooperative edge caching strategies in heterogeneous networks with various edge devices may pose a more significant challenge. Additionally, we will focus on addressing the issue of too many hyperparameter combinations during meta-reinforcement learning training by developing automatic hyperparameter tuning methods.

CRediT authorship contribution statement

Zhenchun Wei: Conceptualization, Writing – original draft, Methodology, Formal analysis. **Yang Zhao:** Validation, Data curation, Software, Visualization. **Zengwei Lyu:** Funding acquisition, Investigation, Resources. **Xiaohui Yuan:** Writing – original draft, Writing – review & editing. **Yu Zhang:** Formal analysis. **Lin Feng:** Project administration, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

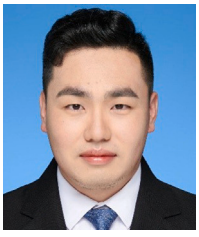
This work was supported by the Natural Science Foundation of Anhui Province, China (2108085MF202), the National Natural Science Foundation of China (62002097), the Fundamental Research Funds for the Central Universities of China (PA2023GDSK0048, PA2023GDGP0044).

References

[1] B. Abolhassani, J. Tadrous, A. Eryilmaz, Optimal load-splitting and distributed-caching for dynamic content over the wireless edge, IEEE/ACM Trans. Netw. 31 (5) (2023) 2178–2190.
[2] U. Cisco, Cisco Annual Internet Report (2018–2023) White Paper, Vol. 10, (1) Cisco, San Jose, CA, USA, 2020, pp. 1–35.
[3] A. Sarah, G. Nencioni, M.M.I. Khan, Resource allocation in multi-access edge computing for 5G-and-beyond networks, Comput. Netw. 227 (2023) 109720.
[4] L. Qiu, G. Cao, Popularity-aware caching increases the capacity of wireless networks, IEEE Trans. Mob. Comput. 19 (1) (2019) 173–187.
[5] M. Reiss-Mirzaei, M. Ghobaei-Arani, L. Esmaeili, A review on the edge caching mechanisms in the mobile edge computing: A social-aware perspective, Internet Things 22 (2023) 100690.
[6] X. Chen, T. Tan, G. Cao, T.F.L. Porta, Context-aware and energy-aware video streaming on smartphones, IEEE Trans. Mob. Comput. 21 (3) (2022) 862–877.
[7] T.X. Tran, A. Hajisami, P. Pandey, D. Pompili, Collaborative mobile edge computing in 5G networks: New paradigms, scenarios, and challenges, IEEE Commun. Mag. 55 (4) (2017) 54–61.
[8] M. Gregori, J. Gómez-Vilardebó, J. Matamoros, D. Gündüz, Wireless content caching for small cell and D2D networks, IEEE J. Sel. Areas Commun. 34 (5) (2016) 1222–1234.

[9] W. Jiang, G. Feng, S. Qin, Optimal cooperative content caching and delivery policy for heterogeneous cellular networks, IEEE Trans. Mob. Comput. 16 (5) (2016) 1382–1393.
[10] J. Kwak, Y. Kim, L.B. Le, S. Chong, Hybrid content caching in 5G wireless networks: Cloud versus edge caching, IEEE Trans. Wireless Commun. 17 (5) (2018) 3030–3045.
[11] Y. Qian, R. Wang, J. Wu, B. Tan, H. Ren, Reinforcement learning-based optimal computing and caching in mobile edge network, IEEE J. Sel. Areas Commun. 38 (10) (2020) 2343–2355.
[12] Q. Li, Y. Sun, Q. Wang, L. Meng, Y. Zhang, A green DDPG reinforcement learning-based framework for content caching, in: 2020 12th International Conference on Communication Software and Networks, ICCSN, 2020, pp. 223–227.
[13] P. Wu, J. Li, L. Shi, M. Ding, K. Cai, F. Yang, Dynamic content update for wireless edge caching via deep reinforcement learning, IEEE Commun. Lett. 23 (10) (2019) 1773–1777.
[14] Y. Zhen, W. Chen, L. Zheng, X. Li, D. Mu, Multiagent cooperative caching policy in industrial internet of things, IEEE Internet Things J. 9 (18) (2022) 16770–16779.
[15] A. Tian, B. Feng, H. Zhou, Y. Huang, K. Sood, S. Yu, H. Zhang, Efficient federated DRL-based cooperative caching for mobile edge networks, IEEE Trans. Netw. Serv. Manag. 20 (1) (2023) 246–260.
[16] Q. Chang, Y. Jiang, F.-C. Zheng, M. Bennis, X. You, Cooperative edge caching via multi agent reinforcement learning in fog radio access networks, in: ICC 2022-IEEE International Conference on Communications, 2022, pp. 3641–3646.
[17] J. Beck, R. Vuorio, E.Z. Liu, Z. Xiong, L. Zintgraf, C. Finn, S. Whiteson, A survey of meta-reinforcement learning, 2023, pp. 1–53, arXiv preprint arXiv: 2301.08028.
[18] Y. Mao, C. You, J. Zhang, K. Huang, K.B. Letaief, A survey on mobile edge computing: The communication perspective, IEEE Commun. Surv. Tutor. 19 (4) (2017) 2322–2358.
[19] M. Zhao, G. Wang, Q. Fu, X. Guo, Y. Chen, T. Li, X. Liu, MW-MADDPG: a meta-learning based decision-making method for collaborative UAV swarm, Front. Neurorobot. 17 (2023) 1–16.
[20] G. Qu, H. Wu, R. Li, P. Jiao, DMRO: A deep meta reinforcement learning-based task offloading framework for edge-cloud computing, IEEE Trans. Netw. Serv. Manag. 18 (3) (2021) 3448–3459.
[21] T. Hospedales, A. Antoniou, P. Micaelli, A. Storkey, Meta-learning in neural networks: A survey, IEEE Trans. Pattern Anal. Mach. Intell. 44 (9) (2021) 5149–5169.
[22] S. Podlipnig, L. Böszörmenyi, A survey of web cache replacement strategies, ACM Comput. Surv. 35 (4) (2003) 374–398.
[23] K. Shanmugam, N. Golrezaei, A.G. Dimakis, A.F. Molisch, G. Caire, Femtocaching: Wireless content delivery through distributed caching helpers, IEEE Trans. Inform. Theory 59 (12) (2013) 8402–8413.
[24] V. Balasubramanian, M. Wang, M. Reisslein, C. Xu, Edge-boost: Enhancing multimedia delivery with mobile edge caching in 5G-D2D networks, in: 2019 IEEE International Conference on Multimedia and Expo, ICME, 2019, pp. 1684–1689.
[25] A. Feriani, E. Hossain, Single and multi-agent deep reinforcement learning for AI-enabled wireless networks: A tutorial, IEEE Commun. Surv. Tutor. 23 (2) (2021) 1226–1252.
[26] Y. He, N. Zhao, H. Yin, Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach, IEEE Trans. Veh. Technol. 67 (1) (2017) 44–55.
[27] N. Garg, T. Ratnarajah, Cooperative scenarios for multi-agent reinforcement learning in wireless edge caching, in: ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP, 2021, pp. 3435–3439.
[28] X. Wu, X. Li, J. Li, P. Ching, V.C. Leung, H.V. Poor, Caching transient content for IoT sensing: Multi-agent soft actor-critic, IEEE Trans. Commun. 69 (9) (2021) 5886–5901.
[29] C. Li, L. Zhu, W. Li, Y. Luo, Joint edge caching and dynamic service migration in SDN based mobile edge computing, J. Netw. Comput. Appl. 177 (2021) 102966–102982.
[30] Y. Zhang, B. Feng, W. Quan, A. Tian, K. Sood, Y. Lin, H. Zhang, Cooperative edge caching: A multi-agent deep learning based approach, IEEE Access 8 (2020) 133212–133224.
[31] M. Radenkovic, V.S.H. Huynh, Cognitive caching at the edges for mobile social community networks: A multi-agent deep reinforcement learning approach, IEEE Access 8 (2020) 179561–179574.
[32] S. Chen, Z. Yao, X. Jiang, J. Yang, L. Hanzo, Multi-agent deep reinforcement learning-based cooperative edge caching for ultra-dense next-generation networks, IEEE Trans. Commun. 69 (4) (2020) 2441–2456.
[33] J. Gao, S. Zhang, L. Zhao, X. Shen, The design of dynamic probabilistic caching with time-varying content popularity, IEEE Trans. Mob. Comput. 20 (4) (2020) 1672–1684.
[34] M.K. Somesula, R.R. Rout, D.V. Somayajulu, Cooperative cache update using multi-agent recurrent deep reinforcement learning for mobile edge networks, Comput. Netw. 209 (2022) 108876–108890.
[35] L. Chen, J. Xu, S. Zhou, Computation peer offloading in mobile edge computing with energy budgets, in: GLOBECOM 2017-2017 IEEE Global Communications Conference, 2017, pp. 1–6.
[36] X. Wang, C. Wang, X. Li, V.C. Leung, T. Taleb, Federated deep reinforcement learning for internet of things with decentralized cooperative edge caching, IEEE Internet Things J. 7 (10) (2020) 9441–9455.
[37] M.T. Spaan, Partially observable Markov decision processes, in: Reinforcement Learning: State-Of-The-Art, 2012, pp. 387–414.
[38] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, I. Mordatch, Multi-agent actor-critic for mixed cooperative-competitive environments, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS, 2017, pp. 6382–6393.
[39] C. Finn, P. Abbeel, S. Levine, Model-agnostic meta-learning for fast adaptation of deep networks, in: International Conference on Machine Learning, PMLR, 2017, pp. 1126–1135.
[40] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, pp. 1–15, arXiv preprint arXiv:1412.6980.
[41] F.M. Harper, J.A. Konstan, The movielens datasets: History and context, ACM Trans. Interact. Intell. Syst. (TIIS) 5 (4) (2015) 1–19.

Zhenchun Wei received the B.S. and Ph.D. degrees from the School of Computer Science and Information Engineering, Hefei University of Technology, Hefei, China, in 2000 and 2007, respectively. He is currently an Associate Professor and M.S. Supervisor with the School of Computer Science and Information Engineering, Hefei University of Technology. His research interests include Internet of Things, distributed intelligence, deep learning, edge computing.



Yang Zhao received his B.E. degree from Hefei University of Technology in 2019. He is currently pursuing the M.S. degree with the School of Computer and Information, Hefei University of Technology, China. His research interests include wireless network optimization and edge caching.

Zengwei Lyu received the B.S. degree in 2012, the M.S. degree in 2015 and the Ph.D. degree in 2020, both from School of Computer Science and Information Engineering, Hefei University of Technology, and he is currently an associate research fellow in Hefei University of Technology. His research fields mainly lie in wireless rechargeable sensor networks.

Xiaohui Yuan received the Ph.D. degree from Tulane University, in 2004. He is currently an Associate Professor with the University of North Texas. His research interests include computer vision, data mining, machine learning, and artificial intelligence. He serves as the chair of several international conferences. He serves on the editorial board of several international journals. He is the Editor-in-Chief of the International Journal of Smart Sensor Technologies and Applications.

Yu Zhang received his B.E. degree from Anhui Medical University in 2021. He is pursuing the postgraduate degree in the School of Computer Science and Information Engineering, Hefei University of Technology, Hefei, China. His research interests include wireless rechargeable sensor networks and deep learning.

Lin Feng received the Ph.D. degree in computer science from the Hefei University of Technology, China, in 2014. She is currently an Associate Professor with the Hefei University of Technology. Her current research interests include vehicular ad hoc networks, wireless networks, and computer vision.