A Multi-Agent Deep Reinforcement Learning-Based Approach to Mobility-Aware Caching

Han Zhao[1], Shiyun Shao[2], Yong Ma[3], Yunni Xia[4(✉)], Jiajun Su[1], Lingmeng Liu[1], Kaiwei Chen[1], and Qinglan Peng[5]

[1] School of Digital Industry, Jiangxi Normal University, Shangrao 334000, China
{zhaohan,sujiajun,202241600165,20214160016}@jxnu.edu.cn
[2] Département d'informatique et recherche opérationnelle, Université de Montréa, Montréal H3T 1N8, Canada
[3] School of Computer and Information Engineering, Jiangxi Normal University, Nanchang 330000, China
may@jxnu.edu.cn
[4] School of Computer Science, Chongqing University, Chongqing 400030, China
xiayunni@hotmail.com
[5] School of Artificial Intelligence, Henan University, Zhengzhou, China
qinglan.peng@henu.edu.cn

Abstract. Mobile Edge Computing (MEC) is a technology that enables on-demand the provision of computing and storage services as close to the user as possible. In an MEC environment, frequently visited content can be deployed and cached upon edge servers to boost the efficiency of content delivery and thus improving user-perceived experience. However, due to the dynamic nature of MEC, it remains a great challenge how to fully exploit mobility information in yielding high-quality content caching decisions for delay-sensitive real-time mobile applications. To address this challenge, this paper proposes a novel mobility-aware caching method by leveraging a Multi-Agent Deep Reinforcement Learning-Based (MAACC) Approach model. The proposed method synthesizes a content fitness algorithm for estimating the priority of caching content with high user fitness and a collaborative caching strategy built upon a multi-agent deep reinforcement learning model. Empirical results clearly show that MAACC outperforms its peers regarding cache hit rate and transfer delay time.

Keywords: Mobile Edge Networks · Cooperative Caching · Content Fit · Multi-agent Deep Reinforcement Learning · Mobility

1 Introduction

The exponential growth of data traffic generated by Internet of Things (IoT) devices, coupled with the increasing number of resource requests from users using

H. Gao et al. (Eds.): CollaborateCom 2023, LNICST 562, pp. 79–95, 2024.
https://doi.org/10.1007/978-3-031-54528-3_5

smart mobile devices (SMD), has posed new challenges to the traditional cloud center model [1]. In the conventional cloud center model, when users request resources from the cloud center, the data needs to be transmitted to the users via backhaul links, resulting in significant content transmission delays that negatively impact the quality of user experience (QoE) [2]. MEC is a technology that extends computing and storage resources to the network edge [3]. It deploys these resources as close to the user's request as possible, typically on edge servers or base stations owned by network operators. As a result, when users initiate requests, the required computing and storage services can be responded to from a location closer to the users, rather than from traditional cloud centers, reducing the distance and time for data transmission.

Despite the advantages of MEC, its highly distributed and dynamic nature, coupled with heterogeneous preferences, service requests, and mobility, presents a challenge in simultaneously ensuring high cache utilization and high user satisfaction. As a result, using reinforcement learning to determine cache strategies has become a popular approach [4]. Reinforcement learning continuously adjusts cache decisions and gradually optimizes cache strategies through interactions with the environment and real-time feedback. It allows for trial-and-error learning during practical execution, identifying effective cache decisions based on received reward signals, and continually refining the strategy to enhance performance. However, in real-world scenarios involving complex interactions among multiple edge servers and mobile users, single-agent reinforcement learning may not effectively capture all variations and interactions, leading to performance limitations [5]. On the other hand, multi-agent reinforcement learning allows each edge server and mobile user to be treated as an independent agent, capable of collaborating or competing [6]. Through interactions with other agents, each individual agent can perceive more environmental information and actions taken by other agents, enabling more accurate decision-making.

Taking all these factors into account, we introduce a new mobile-aware caching approach utilizing multi-agent deep reinforcement learning. This method combines a content fitness algorithm to predict the priority of caching content that suits the user's preferences and a collaborative caching strategy built upon a reinforcement learning model. We conducted extensive simulations to demonstrate its capabilities and advantages over traditional methods.

2 Related Work

In recent years, content caching in the MEC environment has received significant attention as a critical research field both domestically and internationally. This technology is widely recognized as a potential solution for reducing data traffic by deploying popular content locally and thus enabling the server to deliver the content directly to users upon request with greatly reduced latency and network congestion. A series of various MEC caching strategies have been proposed.

Xia *et al.* [7] investigated collaborative caching in edge computing environments, formulated it as a constrained optimization problem, and proposed an

online algorithm for caching decisions. Zhao *et al.* [8] considered MEC caching mechanisms over vehicular networks and aimed at improving the cache hit rate. Recently, personalized caching has also gained considerable attention. Zeng *et al.* [9] proposed a heuristic intelligent caching algorithm that prioritizes content according to user behavioral preferences, based on the historical request count of the corresponding content. Shu *et al.* [10] formulated the decision problem into a collaborative cache optimization model by a collaborative caching (GPCC) method considering group preference and popularity. Then a heuristic algorithm is employed to yield high-quality content placement schedules. Recently, mobility-aware MEC caching drew considerable research interest as well. Musa SS *et al.* [11] provided a mobility-aware active caching solution for an Information Center Network (ICN) generating caching decisions for the maximization of network performance and the reduction of transmission latency. Wei *et al.* [12] predicted the target locations of users and fed predicted trajectories into a caching decision algorithm for yielding cache placement schedules with low response times. Due to the model scale and complexity of the MEC caching problems, deep reinforcement learning algorithms such as Q-learning [13], are recently used and have shown high potency in dealing with related optimization problems. Jiang *et al.* [14] introduced a multi-agent reinforcement learning (MARL) algorithm to address the collaborative content caching problem. The approach is built upon a multi-agent multi-armed bandit model. While the utilization of multi-agent deep reinforcement learning enables experience sharing among agents to enhance learning efficiency, it does not fully take into account the mobility patterns of users, which represents a limitation of the method. Zhong *et al.* [15] presents system models for both centralized caching system and decentralized caching system. They proposed a deep learning-based actor-critic learning framework to optimize content delivery latency. However, unlike this paper, their system models are static in nature. Song *et al.* [16] proposed a single-agent learning mechanism to optimize the cooperative shared caching model for static users in the MEC environment. Compared to single-agent deep reinforcement learning, which often focuses on the learning of a single agent in an isolated environment, multi-agent deep reinforcement learning has the advantage of better leveraging the potential for cooperation among multiple agents. This paper proposes a collaborative caching strategy based on a multi-agent deep reinforcement learning model, which leverages the benefits of collaboration, division of labor, and learning efficiency.

3 System Models

3.1 System Model

In this paper, we configured an MEC environment where one mobile user is connected to one MEC server. The edge computing system models contain caching and mobility two parts. The architecture describes the process of reducing user request latency and optimization to improve cache hit rates in scenarios where users are moving at high speeds.

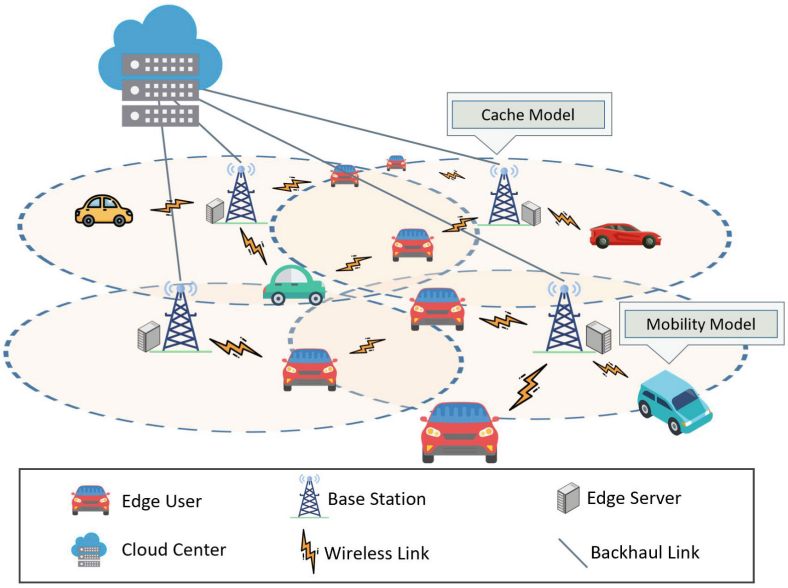


Fig. 1. Edge computing system model.

Each base station in the system (as shown in Fig. 1) is equipped with an edge server that performs computing and caching tasks. The cache size is D and the computing capacity is F (in CPU cycles per second). Edge servers are inter-connected through a wireless network with a bandwidth of ω. Base stations connect to the cloud via a backhaul link. A mobile user first sends a request to the base station, which performs a search in the local cache. If the content is locally available, it is directly sent to the mobile user; otherwise, the base station requests the content from neighboring base stations. If there is still no data, it sends a request to the cloud center.

3.2 Cache Model

In an MEC environment, when users send computation requests, local nodes, edge nodes or cloud servers can all respond and provide required computation services. If a local computation fails to meet the resource request, this request will be forwarded to the nearest server. When both local and edge computations fail to meet, the request is forwarded to the cloud. Handling times of these occasions differs and can be calculated as follows:

– Local calculation:

$$T_{local} = \sum_{i=1}^{m} \sum_{j=1}^{n} \frac{W^{\mathrm{t}}_{bs_i,u_j} \cdot S_f}{C_{bs_i}} \tag{1}$$

where $W^{\mathrm{t}}_{bs_i,u_j}$ denotes the requests generated from the i_{th} base station to the j_{th} user at t time, S_f the size of its request content and C_{bs_i} the processing capacity of the i_{th} base station.

– Edge server calculation:

$$T_{neighbor} = \sum_{i=1}^{m} \sum_{j=1}^{n} \frac{W^{\mathrm{t}}_{bs_i,u_j} \cdot S_f \cdot d_{bs_i,u_j}}{C_{bs_i}} \tag{2}$$

where d_{bs_i,u_j} denotes the distance of communication between the base station and the j_{th} user.

– Cloud server calculation:

$$T_{cloud} = \sum_{i=1}^{m} \sum_{j=1}^{n} \frac{W^{\mathrm{t}}_{bs_i,u_j} \cdot S_f \cdot d_{cloud,u_j}}{C_{bs_i}} \tag{3}$$

where $d_{\mathrm{cloud}\ ,u_j}$ represents the distance between the cloud and the u_{th} user.

In an MEC environment, the mobile devices connect to the edge server through a wireless channel. We define the mobile user u_j to transmit data at a rate of $R^t_{bs_i,u_j}$, to the base station bs_i in time slot t, denoted as:

$$R^t_{bs_i,u_j} = vlog_2(1 + \frac{G_{bs_i,u_j} \cdot P_{bs_i,u_j}}{\sigma^2}),\ 0 < P_{bs_i,u_j} \leq P^{\max}_{bs_i,u_j} \tag{4}$$

where, G_{bs_i,u_j} represents the channel gain from mobile user u_j to base station bsi, measured in dB. σ^2 denotes Gaussian white noise. $Rbs_i, u_j{}^t$ represents the transmission power between mobile user u_j and base station bs_i. The maximum transmission power between mobile user u_j and base station bsi is defined as $Pbs_i, u_j{}^{\max}$. These parameters play a crucial role in the communication process.

Therefore, the communication time between mobile user u_j to base station bs_i is expressed as:

$$T_{comm} = R^t_{bs_i,u_j} \cdot d_{bs_i,u_j}, 0 < d_{bs_i,u_j} \leq d_{\max} \tag{5}$$

The time delay incurred by the mobile user when requesting content depends on the location of the requested content, local base station, nearby base station or appearance time in the cloud. The delay time is as follows:

– Local Caching: The time delay is considered to be 0 when the cached content is available to be requested at the local base station.

$$T\_dy_{local} = 0 \tag{6}$$

– Edge Cooperation: When the cached content does not exist at the local base station but can be requested at the neighboring base station, the transmission delay time can be expressed as follows, assuming the base station has the same transmit power:

$$T\_dy_{neighbor} = \sum_{i=1}^{m} \sum_{j=1}^{n} \frac{W^{\mathrm{t}}_{bs_i,u_j} \cdot S_f \cdot \theta_w}{R^t_{bs_i,u_j}} \tag{7}$$

where θ_w represents the ratio between the amount of data contained in the request W_{bs_i,u_j} sent by the user u_j to the base station bs_i at time slot t and the amount of data returned by the base station in response to the request.

– Cloud Fetching: When a user requests resources cached in the cloud, the content must be transmitted to the user through a backhaul link, leading to an increase in time delay.

$$T\_dy_{cloud} = \sum_{i=1}^{m} \sum_{j=1}^{n} \frac{W_{bs_i,u_j}^{\mathrm{t}} \cdot S_f \cdot \theta_w}{R_{cloud}} \tag{8}$$

where R_{cloud} represents the transmission rate of the backhaul link between the cloud and the base station.

3.3 Mobility Model

Assume that the mobility of a mobile user follows an arbitrary pattern, where the direction and angle of movement are time-varying. Thus the user's path is expressed in terms of latitude and longitude as follows:

$$L_{u_i}^t = \{lon_{u_i}(t), lat_{u_i}(t)\} \tag{9}$$

$L_{u_i}^t$ denotes the moving trajectory of the i_{th} user in time slot t. Where $lat_{u_i}(t)$ denotes the longitude point of the mobile trajectory of the i_{th} user in time slot t, and $lon_{u_i}(t)$ denotes the latitude point of the mobile trajectory of the i_{th} user in time slot t.

4 Cache Scheme

4.1 Problem Formulation

In MEC-based network architectures, where users are moving at high speed and placed in the cache using various policies, our objective is to minimize the content delivery latency. Therefore, the caching problem can be formulated as an optimization process with the following goal:

$$\begin{aligned} \mathbf{F1}: T_{\min} = \min_{s} \frac{1}{m}\frac{1}{n} \sum_{i=1}^{m}\sum_{j=1}^{n} \alpha_1 \left(\lambda_1 T_{\text{local}} + \lambda_2 T_{\text{neighbor}} + \lambda_3 T_{\text{cloud}}\right) + \alpha_2 T_{\text{comm}} \\ + \alpha_3 \left(\eta_1 T\_dy_{\text{local}} + \eta_2\ T\_dy_{\text{neighbor}} + \eta_3 T\_dy_{\text{cloud}}\right) \end{aligned} \tag{10}$$

$$\begin{aligned} s.t.\quad & \mathbf{C1.}\ \textstyle\sum_{j=1}^{n} W_{u_j}^t \cdot S_f \le D \quad \forall\ j \in [1,\ n] \\ & \mathbf{C2.}\ T\_dy \le T\_dy_{max} \\ & \mathbf{C3.}\ 0 < d_{u_j} \le d_{max} \qquad \forall\ j \in [1,\ n] \end{aligned}$$

where, $\lambda_1, \lambda_2, \lambda_3$ denote the weight respectively in the local base station, proximity base station, and computation time delay of cloud generation; η_1, η_2, η_3 denote the weight when the local base station, proximity base station, and transmission time delay of content generation in the cloud; $\alpha_1, \alpha_2, \alpha_3$ denote the weight when the computation model generates time delay and the weight when the content transmission generates time delay. The weight value constraints are

$$\begin{array}{ll} \textbf{(a)}\ \lambda_1 + \lambda_2 + \lambda_3 = 1 & \lambda_1, \lambda_2, \lambda_3 \in (0,1) \\ \textbf{(b)}\ \eta_1 + \eta_2 + \eta_3 = 1 & \eta_1, \eta_2, \eta_3 \in (0,1) \\ \textbf{(c)}\ \alpha_1 + \alpha_2 + \alpha_3 = 1 & \alpha_1, \alpha_2, \alpha_3 \in (0,1) \end{array}$$

The set s represents all rational caching strategies. Constraint C1 ensures that the size of the content requested by the user cannot exceed the capacity size of the edge server. Constraint C2 represents a delay time duration constraint that ensures that the upper limit of requested time is the maximum allowable delay time. Constraint C3 indicates the maximum distance of the user request content.

4.2 Multiple DRL Agents

The idea behind reinforcement learning is based on mutual learning between an intelligent agent and its environment. Through continuous attempts, the agent learns to determine the best behavior. In this paper, we use a multi-intelligent deep reinforcement learning framework, and partially incomplete observable reinforcement learning problems can be modeled as Partially Observable Markov Decision Processes (POMDP) [18] modeled as a six-tuple $\{S, A, R, P, \zeta, \delta\}$. S represents the set of states, A denotes the set of actions, R represents the reward function, and P corresponds to the state transition probability matrix.

$$P = \begin{bmatrix} P_{11} & \cdots & P_{1n} \\ \vdots & \ddots & \vdots \\ P_{n1} & \cdots & P_{nn} \end{bmatrix}$$

In this context, ζ refers to the set of observations, while δ represents the set of observation probabilities. The state, action, and reward can be defined as follows:

1) State: The state space reflects the state of the mobile user's environment. In multi-agent systems, the set of the proxy is defined as $Ag = \{ag_1, ag_2, ..., ag_n\}$, $ag_i = \{info_i, neighbor_i, CN_i\}$, $info_i$ refers to the content caching status of agent i, $neighbor_i$ signifies the content storage status of neighboring nodes for agent i, CN_i represents the predicted content for the next moment, acquired using **Algorithm** 1. Each agent determines its own independent caching decisions through actions and global state Π_{ag_i} and estimate the strategy of other agents Π_{ag_j}, the global optimal strategy Π is finally determined.

2) Action: The action set represents the current edge server's decision on whether to cache the content fitness resources or not. Since the number of users served by each base station is variable, the number of resources arriving at the base

station in the same time slot varies, and consequently, the action space size of each agent also varies. Notation of the action set as $AC = \{Ac_c^t, Ac_r^t\}$, where Ac_c^t represents the caching action, $Ac_c^t = \{Ac_{c1}^t, Ac_{c2}^t, ..., Ac_{cr}^t\}$, Ac_{ci}^t represents whether the current base station caches the i_{th} content in CN; Ac_r^t represents the replacement action, the content should be replaced with the content of the resource with lower content fitness.

3) Reward: The reward function is primarily designed to minimize content delivery latency T_{min}, and the reward value increases when the action satisfies the request.

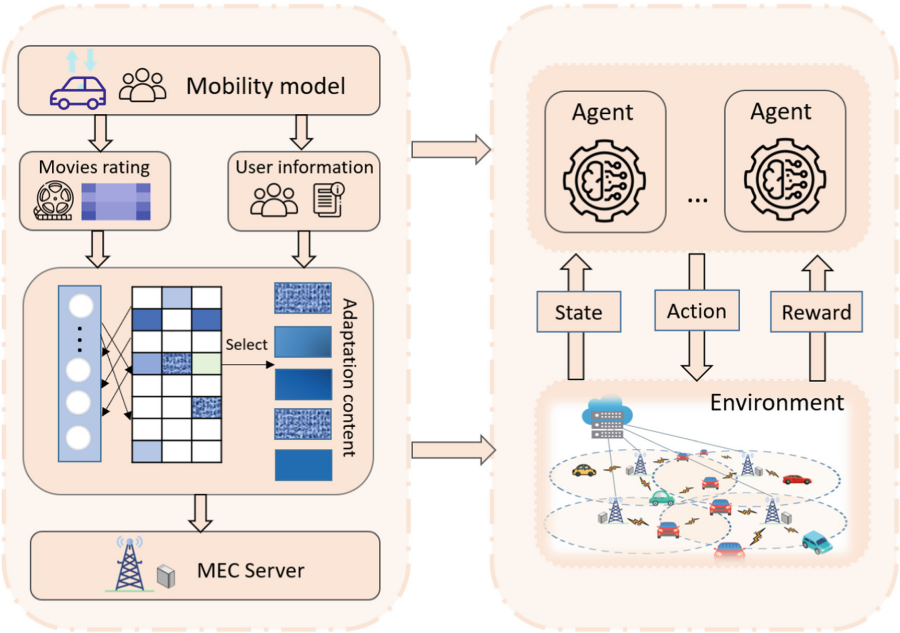


Fig. 2. Caching behavior of edge servers when users move.

4.3 Adaptation Resource Forecast

Content fit is defined as the degree to which content is suitable for the user, serving as a metric to assess the alignment between content and user demands. A higher content fit indicates that the cached content better matches the user's requests, thus enhancing user experience and system performance. Based on the characteristics of user requests, we outline the features of user-requested

content and propose a predictive method for content fit. The detailed algorithm is presented in Algorithm 1.

Algorithm 1. Content fit algorithm

K : Content type
$T = \{t_0, t_1, ..., t_n\}$: Time period T is then divided into n time slots
$f_k\,(t)$: Access content
$R_{f_k}(t_m)$: Number of requests for the corresponding content
$P_{f_k}^{t_m}$: Content fitness
$Cha_k(t_m)$: Retrieve the relevant attributes of the content
S_k: Content feature adaptation degree
Collect historical request data from users during time period T, including user IDs, timestamps, and requested content information.
for each $u_i \in$ Users **do**
    $P_{f_k}^{t_\mathrm{m}} = \frac{R_{f_k}(t_\mathrm{m})}{\sum_k R_{f_k}(t_\mathrm{m})}$
    Predict $\hat{P}_{f_k}^{t_m+1}$ from $\left\{P_{f_k}^{t_0}, \quad P_{f_k}^{t_1}, \ldots, P_{f_k}^{t_\mathrm{m}}\right\}$
    $\hat{P}_{f_k}^{t_m+1}$ for $f_k\,(t_m+1)$
    Predict $\widehat{f_k}(t_\mathrm{m}+1)$
    Predict $\widehat{\mathrm{Cha}_k}(t+1)$ from $\widehat{f_k}(t_\mathrm{m}+1)$
    Calculate $S_k = \sqrt{\sum_k \theta_k(\widehat{Cha_k}(t_m+1) - Cha_k(t_m))^2}$
    Calculate $P_{f_k}(t_m+1) = \frac{\sum_{i=0}^{N} S_{k,i}, P_{f_{k,i}}^{t_m}}{N}$
    Uploads $P_{f_k}^{t_m}$ to the local BS
end for
Content suitability assessment and selection for inclusion in CN.
return CN

The content fitness $P_{f_k}^{t_m}$ is defined as the proportion of requests for a specific content at time slot t_m to the total number of user requests. It reflects the proportion of this content in user requests, and a higher content fitness value indicates that the content is more popular among users, as it occupies a larger proportion in user requests. Useful content features are extracted from historical request data and cache content properties, denoted as $Cha_k(t_m)$. These features mainly include content characteristics preferred by users, such as content type and size. S_k represents the predicted similarity of content features, helping to predict the similarity between different content features by calculating a similarity score between them. The relevant formulas are shown in Algorithm 1.

The entire process can be summarized as follows: By predicting the content suitability value for the next moment. $\widehat{P}_{f_k}^{t+1}$, predicting the characteristics of the next moment of content $\widehat{\mathrm{Cha}}_k(t+1)$, the predicted content for the next moment is obtained by calculation $\widehat{f_k}\,(t+1)$ and the fitness of the content $f_k\,(t)$ content suitability for the next moment can be determined $P_{f_k}^{t+1}$, from there, the corresponding content can be determined $f_k\,(t+1)$.

4.4 Multi-Agent Actor-Critic Algorithm for Content Caching

Algorithm 2 is known as MAAC-based Cooperative Cache Algorithm (MAACC). When mobile users send requests, the MEC node simultaneously receives the requests and their features. It then provides the current cache status to the actor network, allowing cache operations to be obtained within the current time slot. After executing actions based on the policy, each agent receives rewards and the next state, and the information received from the environment is stored as a history for future reference. The decision process of the caching policy is shown in Fig. 2: 1) Users move according to the model and send requests and interested content to the MEC node, where the interested content is predicted by Algorithm 1; 2) The system state is observed, and actions are guided according to the corresponding rewards; 3) Based on reward signals and the current system state, the algorithm generates the final collaborative caching decision, guiding cache operations on edge servers to maximize cache hit rate and user satisfaction.

Algorithm 2. MAACC algorithm

Input: Number of iterative rounds $T, t = 1$, number of steps per turn, learning rate δ, discount Factor γ, actor-critic network structure.

Initialization: Initialize system parameters, hyper parameters, edge server cache space, mobile user cache space, and initialize a random process for action exploration.
for each episode **do**
  for each $t < T$ **do**
    MEC receives user requests
    Observe the state of this cache for each agent
    Input all the states into the actor network to get action
    Choose the right action for each agent
    Execute the action to store the received reward value and the new status information in the history
    $t = t + 1$
  end for
  Store the event set for each agent
  update the target network using the formula
  Update content properties and cache state
  Update network parameters
end for

Output: Optimal strategy Π

5 Performance Evaluation

5.1 Parameters Setting

We perform simulations and compare the proposed approach with other existing solutions. The simulation configuration utilized the *Shanghai Telecom*

dataset [19], which comprises more than 7.2 million records of content access events recorded from 3,233 edge stations and 9,481 mobile users over a span of six months, along with corresponding mobile traces. *Movielens 1M* [20] dataset simulates the content requests from different mobile users. Randomly match a user trajectory from the Shanghai Telecom dataset to each user in the Movielens 1M dataset. Figure 3 shows the distribution location of edge nodes in Shanghai. Figure 4 is an example recording the trajectory of a taxi in the city heart of Shanghai. We use Python to implement the proposed MAACC method. The relevant parameters are shown in Table 1.

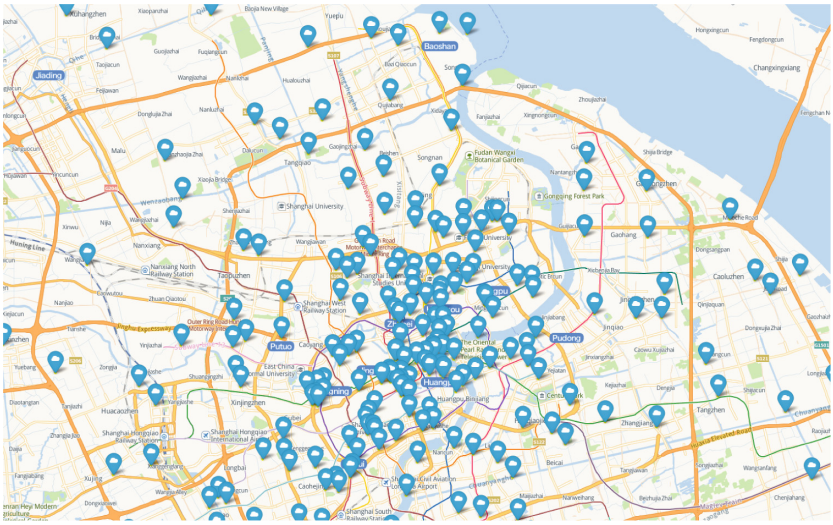


Fig. 3. Blue coordinate icons represent each edge node in Shanghai's downtown area showing the general distribution of the whole network. (Color figure online)

5.2 Comparison Algorithms

MAACC is compared against the following benchmark algorithms: Thompson sampling (TS) [21], Random Selection Algorithm (RSA) [22], Greedy Algorithm (GA) [23], and the Multi-Agent caching strategy MARL [14].

5.3 Performance Analysis

For performance analysis, we set up an environment where each base station was equipped with an edge server. Different base stations cached corresponding contents according to corresponding caching strategies to test the resource requests of mobile vehicles.

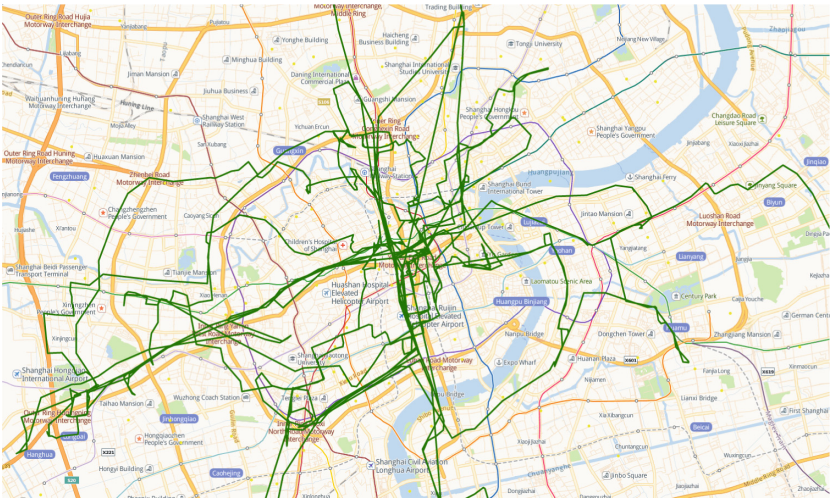
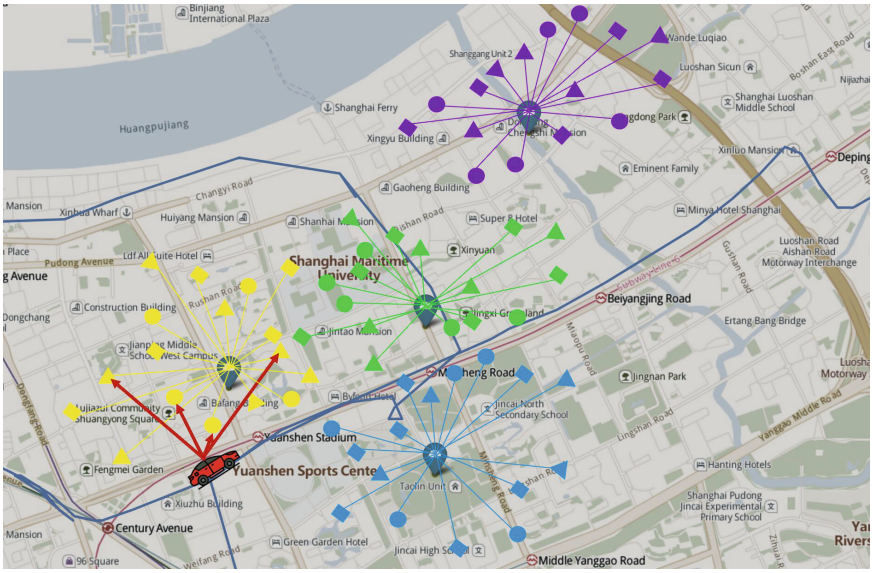


Fig. 4. The green lines represent the route trajectory of a taxi in the city heart of Shanghai on a certain day. (Color figure online)

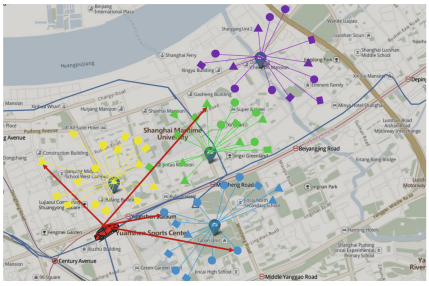
Table 1. Parameter table

Parameter	Value
Number of users	30
Coverage radius (m)	100–200
Basic Transfer Data Size (KB)	8
Basic run time per task (ms)	20
Total bandwidth (Mbps)	50
Number of rounds of training	30
Size of cache	0–500
The number of local epochs	10
Local batch size	50
Actor and critic learning rate	0.001, 0.0005
Network update rate	0.01
Discount	0.9

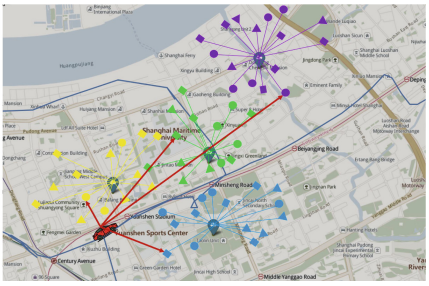
As shown in Fig. 5, different kinds of applications are represented by different shapes, such as triangles, prototypes and squares. Different colors indicate the resources cached by different edge servers. When the vehicle moves, the resource hit rates of MAACC and MARL cache are higher than others, which means the vehicle can obtain resources from the nearest base station. On the contrary, TS/RSA/GA request resources from adjacent edge servers resulting in increased latency.



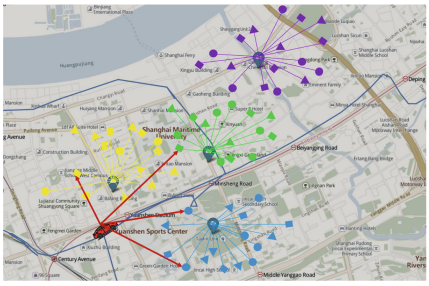
(a) MAACC



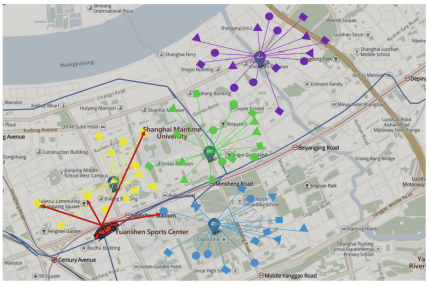
(b) TS



(c) RSA



(d) GA



(e) MARL

Fig. 5. Cached resources using different algorithms.

Figure 6 displays the cache hit rates for various caching strategies with different cache capacities. With the increase in edge server capacity, the cache hit ratio also rises. A larger cache capacity allows for more resources to be cached, which increases the likelihood of mobile users obtaining resources from local and nearby edge servers. This results in a higher hit ratio. And MAACC beats TS/RSA/GA/MARL by 22.2%/37.6%/15.2%/12.8%, respectively.

Figure 7 reveals the request latency for different caching strategies with different cache capacities. Among them, the MAACC method has lower latency compared to other methods. As the cache capacity of the edge server increases, the resource transfer latency for all cache strategies decreases. Indeed, the larger cache capacity contributes to a higher cache hit ratio for the edge server, increasing the likelihood of mobile users obtaining resources from local and neighboring servers. As a result, the resource transmission delay is reduced, leading to improved performance in delivering content to users.

Figure 8 compares the cache hit rates of MAACC and MARL in different rounds when the edge server capacity is 300. Within 30 rounds, the MAACC scheme achieves an average cache hit rate of 54.02%, which is 22.85% higher than the MARL scheme. This indicates that the proposed MAACC scheme in this paper slightly outperforms the MARL scheme, and the reason for the improved cache hit rate is that the MAACC scheme takes into account the mobility of users and selects suitable content for caching.

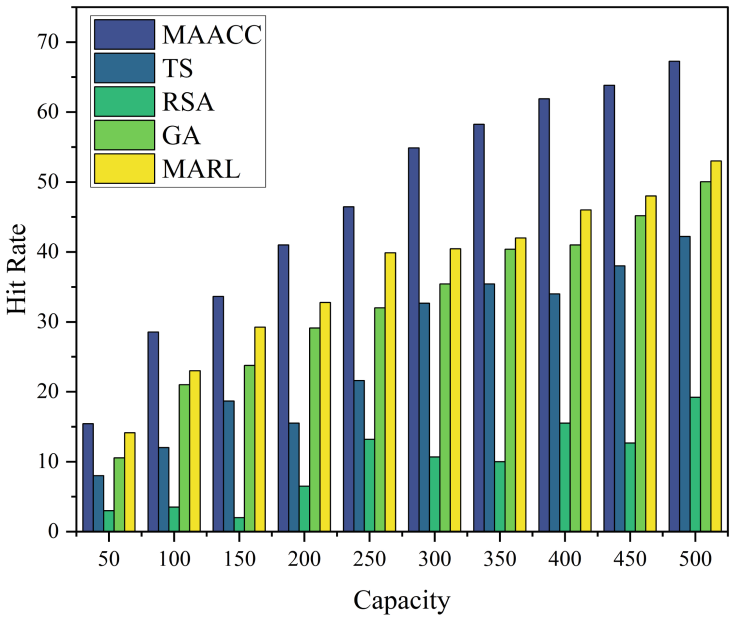


Fig. 6. Cache hit rates of MAACC/TS/RSA/GA/MARL under different cache capacities.

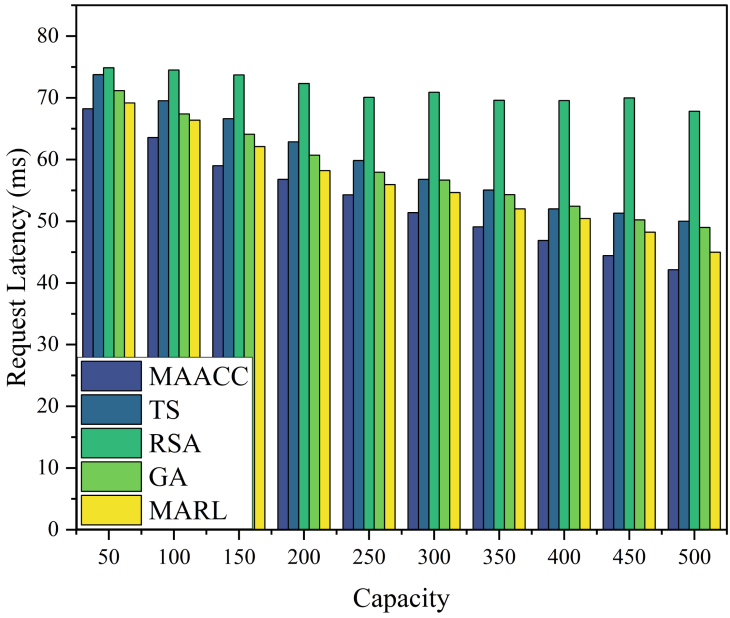


Fig. 7. Request latency of MAACC/TS/RSA/GA/MARL under different cache capacities.

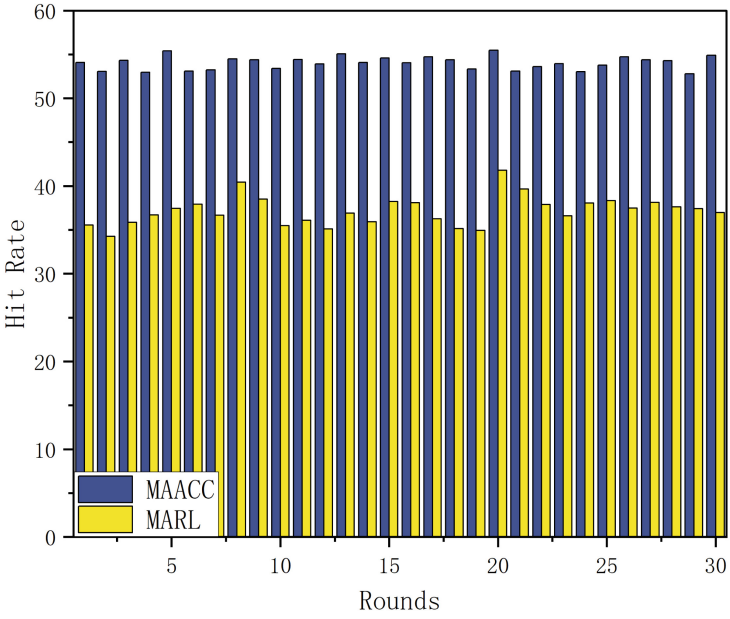


Fig. 8. Hit rate of MAACC and MARL under identical capacities but different rounds.

6 Conclusion

This paper primarily investigates the challenge of high-speed moving resource caching in the MEC environment and proposes a collaborative caching scheme MAACC based on a multi-agent deep reinforcement learning algorithm. Experimental results demonstrate that this method effectively improves cache hit rate and reduces content transmission delay. Moving forward, we intend to explore the impact of edge node failures during resource caching and integrate a checkpoint algorithm to proactively remove untrusted edge nodes, aiming to further optimize the caching strategy in MEC environments.

Acknowledgment. This article is supported by the Innovation Fund Project of Jiangxi Normal University(YJS2022065) and Domestic Visiting Program of Jiangxi Normal University. Additionally, this work is supported in part by Henan Province Science and Technology Projects (232102210024).

References

1. Farooq, M.J., Zhu, Q.: A multi-layer feedback system approach to resilient connectivity of remotely deployed mobile internet of things. IEEE Trans. Cogn. Commun. Networking **4**(2), 422–432 (2018)
2. Cao, K., Liu, Y., Meng, G., Sun, Q.: An overview on edge computing research. IEEE Access **8**, 85714–85728 (2020)
3. Chen, Z., Chen, Z., Ren, Z., Liang, L., Wen, W., Jia, Y.: Joint optimization of task caching, computation offloading and resource allocation for mobile edge computing. China Commun. **19**, 142–159 (2022)
4. Qiao, G., Leng, S., Maharjan, S., Zhang, Y., Ansari, N.: Deep reinforcement learning for cooperative content caching in vehicular edge computing and networks. IEEE Internet Things J. **7**(1), 247–257 (2020)
5. He, Y., Yu, F.R., Zhao, N., Leung, V.C.M., Yin, H.: Software-defined networks with mobile edge computing and caching for smart cities: a big data deep reinforcement learning approach. IEEE Commun. Mag. **55**(12), 31–37 (2017)
6. Wang, R., Li, M., Peng, L., Hu, Y., Hassan, M.M., Alelaiwi, A.: Cognitive multi-agent empowering mobile edge computing for resource caching and collaboration. Future Gener. Comput. Syst. **102**, 66–74 (2020)
7. Xia, X., Chen, F., He, Q., Grundy, J., Abdelrazek, M., Jin, H.: Online collaborative data caching in edge computing. IEEE Trans. Parallel Distrib. Syst. **32**(2), 281–294 (2021)
8. Zhao, J., Sun, X., Li, Q., Ma, X.: Edge caching and computation management for real-time internet of vehicles: an online and distributed approach. IEEE Trans. Intell. Transp. Syst. **22**(4), 2183–2197 (2021)
9. Zeng, Y., et al.: Smart caching based on user behavior for mobile edge computing. Inf. Sci. **503**, 444–468 (2019)
10. Yao, T., Chai, Y., Wang, S., Miao, X., Bu, X.: Radio signal automatic modulation classification based on deep learning and expert features. IEEE Xplore (2020)
11. Musa, S.S., Zennaro, M., Libsie, M., Pietrosemoli, E.: Mobility-aware proactive edge caching optimization scheme in information-centric IoV networks. Sensors **22**(4), 1387 (2022)

12. Wei, H., Luo, H., Sun, Y.: Mobility-aware service caching in mobile edge computing for internet of things. Sensors **20**(3), 610 (2020)
13. Sadeghi, A., Sheikholeslami, F., Giannakis, G.B.: Optimal and scalable caching for 5g using reinforcement learning of space-time popularities. IEEE J. Sel. Topics Signal Process. **12**(1), 180–190 (2018)
14. Jiang, W., Feng, G., Qin, S., Liang, Y.-C.: Learning-based cooperative content caching policy for mobile edge computing. In: ICC 2019–2019 IEEE International Conference on Communications (ICC). IEEE (2019)
15. Zhong, C., Gursoy, M.C., Velipasalar, S.: Deep reinforcement learning-based edge caching in wireless networks. IEEE Trans. Cogn. Commun. Networki. **6**(1), 48–61 (2020)
16. Song, J., Sheng, M., Quek, T.Q.S., Xu, C., Wang, X.: Learning-based content caching and sharing for wireless networks. IEEE Trans. Commun. **65**(10), 4309–4324 (2017)
17. Jeong, S., Simeone, O., Kang, J.: Mobile edge computing via a UAV-mounted cloudlet: optimization of bit allocation and path planning. IEEE Trans. Veh. Technol. **67**(3), 2049–2063 (2018)
18. Cassandra, A.R., Littman, M.L., Zhang, N.L.: Incremental pruning: a simple, fast, exact method for partially observable Markov decision processes. arXiv:1302.1525 cs (2013)
19. Li, Y., Zhou, A., Ma, X., Wang, S.: Profit-aware edge server placement. IEEE Internet Things J. **9**(1), 55–67 (2022)
20. Harper, F.M., Konstan, J.A.: The MovieLens datasets. ACM Trans. Interact. Intell. Syst. **5**(4), 1–19 (2015)
21. Cui, L., et al.: CREAT: blockchain-assisted compression algorithm of federated learning for content caching in edge computing. IEEE Internet Things J. **9**(16), 14151–14161 (2022)
22. Xiao, H., Zhao, J., Pei, Q., Feng, J., Liu, L., Shi, W.: Vehicle selection and resource optimization for federated learning in vehicular edge computing. IEEE Trans. Intell. Transp. Syst. **28**, 11073–11087 (2021)
23. Banerjee, B., Kulkarni, A., Seetharam, A.: Greedy Caching: an optimized content placement strategy for information-centric networks. Comput. Networks **140**, 78–91 (2018)