Peer-to-Peer Networking and Applications (2025) 18:93
https://doi.org/10.1007/s12083-025-01915-y

Service caching with multi-agent reinforcement learning in cloud-edge collaboration computing

Yinglong Li[1] · Zhengjiang Zhang[1] · Han-Chieh Chao[2,3,4,5,6]

Received: 8 November 2024 / Accepted: 25 January 2025 / Published online: 10 February 2025

Abstract

Edge computing moves application services from the central cloud to the network edge, significantly reducing service latency. Edge service caching presents a more complex challenge than cloud caching, due to the dynamics and diversity of mobile user requests. Consequently, traditional caching strategies are not directly applicable to edge environments. Additionally, the challenge intensifies when considering collaborative caching between adjacent servers. To address these challenge, we propose an edge service caching solution aimed at minimizing the total service delay to ensure high quality user experiences. First, given the limited prior information on user requests in the current time period, we adopt a Transformer-based approach to enhance the accuracy of user request predictions. Since the service caching problem involves both continuous and discrete action spaces, we propose a deep reinforcement learning algorithm based on hybrid Soft actor-critic (SAC) to learn the optimal caching strategy. We then leverage a centralized training and decentralized decision making framework to address multi-agent problems, while selectively reducing agent observation connections to avoid the interference from redundant observations. Finally, extensive simulations demonstrate that our proposed collaborative cloud-edge service caching strategy reduces service latency more effectively than existing approaches.

Keywords Edge computing · Service caching · Resource allocating · Multi agent reinforcement learning

This article is part of the Topical Collection: *1- Track on Networking and Applications*
Guest Editors: Vojislav B. Misic

✉ Zhengjiang Zhang
zhangzhenjiang@bjtu.edu.cn

Yinglong Li
ylongli@bjtu.edu.cn

Han-Chieh Chao
hcc@ndhu.edu.tw

1 School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing 100044, China

2 Tamkang University, Taipei 251, Taiwan

3 Chair Professor, Department of Applied Informatics, Fo Guang University, Yilan, Taiwan

4 Distinguished Chair Professor, Department of Artificial Intelligence, Tamkang University, New Taipei, Taiwan

5 Emeritus Chair Professor, National Dong Hwa University, Hualien, Taiwan

6 Distinguished Visiting Professor, UCSI, Kuala Lumpur, Malaysia

1 Introduction

The development of wireless communication and artificial intelligence has led to a rapid increase in network devices, including mobile phones, computers, smart wearable devices and so on. These devices connect to the internet to access a range of services, such as autonomous driving, content access, intelligent recognition, and voice analysis, most of which are computing-intensive [1]. To process the service requests of end users, service providers usually cache the corresponding services on the central cloud to respond to end-user requests in a wide range [2]. However, as application services diversify, cloud computing is increasingly unable to deliver high-quality user experiences.

Centralized cloud computing faces several key challenges: 1) A large number of devices requesting services put significant pressure on cloud center bandwidth, leading to severe network congestion [3]. Additionally, the cloud center's computing resources are insufficient to support numerous intelligent services. 2) The long distance between users and the central cloud leads to high transmission delays, making it difficult for cloud computing to meet the low-latency

demands of delay-sensitive services [4]. Edge computing [5, 6], a new computing paradigm, can effectively address these challenges. Edge computing deploys servers near users to process tasks at the network's edge, effectively reducing transmission delays associated with cloud computing. By deploying multiple servers at the network edge, efficiently scheduling user requests, and deploying services, edge computing not only alleviates bandwidth pressure but also delivers computing services more efficiently.

In practice, the central cloud is located farther from users than edge servers, resulting in significantly higher response delays compared to edge servers. To minimize response delays, service providers should cache services on edge servers whenever possible. Due to limited computing resource and storage capacity, edge servers cannot feasibly cache all services. An optimal caching strategy is needed to efficiently utilize edge servers' limited storage capacity, reducing response delays for user service requests. Some existing approaches predict the popularity of user-requested services based on historical data and known service popularity models. However, these models fail to address the time-based dynamics and environmental heterogeneity of user requests in edge service caching.

The edge caching problem presents three key challenges. 1) Most existing works use service caching strategies with fixed computing resource allocations, rarely exploring dynamic allocation by edge servers. These static strategies fail to adapt to the heterogeneity of edge servers, so we propose a hybrid variable caching strategy that integrates service caching and dynamic resource allocation. 2) Mainstream research typically relies on historical data to predict the popularity of user service requests or assumes service popularity is known, but this is impractical. Modern services are diverse, and user requests exhibit dynamic, time-dependent correlations. We explored a Transformer-based method to capture these dynamics and correlations over time. 3) Given the variability of user service requests across different scenarios, we explored deep reinforcement learning (DRL) [7, 8] to address edge service caching across multiple scenarios. However, traditional DRL typically employs centralized learning for service caching, which is unsuitable for large-scale edge caching problems due to the vast action space created by numerous edge servers. Therefore, we adopted multi-agent reinforcement learning (MARL) [9, 10] with a distributed service caching strategy, allowing each server to implement its own decision-making approach. Considering the hybrid action space in our problem-comprising both the discrete service cache action space and the continuous service resource allocation action space-we propose a multi-agent hybrid SAC (Soft Actor-Critic) deep reinforcement learning model. This model integrates the Transformer's predictions as part of each server's observation value and uses its own policy network to decide on service cache actions.

The key contributions of our paper are summarized below:

We establish an edge service caching system based on cloud-edge collaboration within the edge computing network. From the user's perspective, we consider the delay in responding to service requests as a system cost. We further model this problem as a Markov process to minimize the expected system cost.

In diverse heterogeneous scenarios, we propose a multi-agent hybrid soft actor-critic algorithm where all agents share a critic network to learn service request heterogeneity. Each agent concurrently trains two actor networks to learn discrete and continuous actions, while adjacent agents share their state values to enhance cooperation. Additionally, each agent employs a Transformer network to capture the time series correlations of user service requests, improving the prediction of requested service types.

We perform extensive experiments on simulated datasets within controlled environments. The training results indicate that the proposed algorithm outperforms other reinforcement learning algorithms in terms of convergence. Moreover, regarding service caching decisions, it significantly reduces the response delay for user service requests.

The rest of this paper is structured as follows. In Section 2, we review the related works on DRL-based service replacement and multi-agent reinforcement learning. Section 3 introduces the system architecture, latency model, and problem formulation. Section 4 presents multi-agent based on transformer hybrid soft actor-critic algorithm to address service caching problem. The simulation results and analysis are presented in Section 5. We draw a conclusion of this paper in Section 6.

2 Related work

2.1 DRL-based service replacement

Most of the existing studies on edge service caching strategies are aimed at user service quality or system cost. [11] Zhou et al. [12] proposed a centralized task offloading and resource allocation method to solve the service caching joint optimization problem. He used a centralized method to determine the service caching strategy with the goal of minimizing system cost. To solve the service caching problem in edge networks, Xu et al. [13] proposed a random rounding method to solve the integer programming problem, and then used game theory to minimize the cost of system. However, with the increasing number of application services, traditional heuristic methods are difficult to solve the service caching problem with an increasingly large solution space. With the popularization of artificial intelligence, studies have found that deep reinforcement learning [14–16] is capable of handling large-scale service caching problems. Chen et al. [17] used DRL to solve

the joint problem of service caching and computing, and proposed a double-delayed deep deterministic policy gradient training method. Xue et al. [18] studied the service caching and computing offloading problems of time-varying requests in vehicular edge networks with highly dynamic topologies, and proposed a DRL algorithm to solve mixed integer nonlinear programming problems. Zhou et al. [19] studied the service caching, task scheduling and resource allocation problems of multiple users' collaboration. With the goal of minimizing system energy consumption, an algorithm based on deep deterministic policy gradient was proposed. Tang et al. [20] developed an actor-critic deep reinforcement learning algorithm to address the joint service deployment and task scheduling problem. In this study, user requests are assumed to be random, and the popularity of requested services is not explicitly considered. Our approach employs a transformer to learn the popularity patterns of user service requests and capture the temporal correlations between them. Wei et al. [21] propose a deep reinforcement learning-based algorithm to address the joint optimization problem of UAV trajectory planning, task scheduling, and service deployment. This paper focuses on independent task scenarios and does not apply to multi-task scenarios. Our approach leverages collaboration between edge servers to tackle the service deployment challenge in various multi-task environments.

2.2 Multi-agent reinforcement learning

Many current edge computing problems are being studied using deep reinforcement learning, such as computing offloading [22], service deployment [23], and resource scheduling [24], etc. However, the diverse computing and storage capabilities of each edge computing server and the different user preferences they serve result in poor results for centralized deep reinforcement learning algorithms. Multi-agent deep reinforcement learning [25] is proposed to solve these problems. Zhong et al. [26] proposed a service caching algorithm based on multi-agent actor-critic deep reinforcement learning to minimize system transmission delay when the popularity of user preferences is unknown. Zhou et al. [27] studied the service deployment strategy considering service distribution and replacement, and proposed a reinforcement learning algorithm based on multi-agent collaboration to minimize the cost of service, in which each edge server makes the best local decision. Yao et al.[28] studied the joint problem of task scheduling and service caching based on the cooperation of edge servers, and proposed a MARL algorithm combined with graph attention network to learn the joint strategy. The graph attention network is used to learn the relationship between edge servers. Huang et al. [29] studied the dynamic service deployment strategy with the goal of minimizing the service provider cost. A reinforcement learning dynamic service deployment algorithm based on multi-agent collaboration is proposed. Hu et al. [30] proposed a cloud-edge collaborative architecture for multi-objective optimization and cache popularity prediction, along with a horizontal and vertical collaborative cache decision strategy. However, this approach did not investigate its applicability in multi-heterogeneous content scenarios. Wu et al. [31] proposed a multi-agent edge collaboration architecture to address the multi-objective active caching problem and introduced a deep Q-network method. However, the deep Q-network method suffers from limited exploration and unstable training. In contrast, the SAC algorithm we propose improves action exploration and enhances sample efficiency by incorporating entropy regularization. Lin et al. [32] proposed a user preference-aware content caching and migration scheme and developed a multi-agent reinforcement learning algorithm for decision-making. However, this algorithm is limited to discrete decision actions and does not account for continuous actions. Our proposed algorithm addresses this limitation by training two actors on each agent to separately learn discrete and continuous actions.

3 System model and problem formultion

In this section, we first introduce the system architecture for service caching based on cloud-edge collaboration, including the network model and latency model. We then formulate the objective problem.

3.1 System architecture

As illustrated in Fig. 1, we consider a cloud-edge collaborative computing network comprising three entities: a central cloud, the edge layer and the user layer. The central cloud is equipped with ample computational resources and caches all the application services required by end users. The edge layer consists of multiple edge servers, each equipped with resources for service caching and communicating through base stations. The user layer is composed of end users, categorized by geographical contexts such as residential, corporate, and educational settings, with each area served by a specified number of edge servers. Typically, a user's service request is processed by the nearest server. However, if the requested service is not cached locally, the request is forwarded to a nearby edge server or the central cloud. In this network, edge servers can cache various application services from the central cloud based on user service requests. The service requests of users in different edge scenarios are heterogeneous, leading to heterogeneous distributions of cached application services among edge servers.

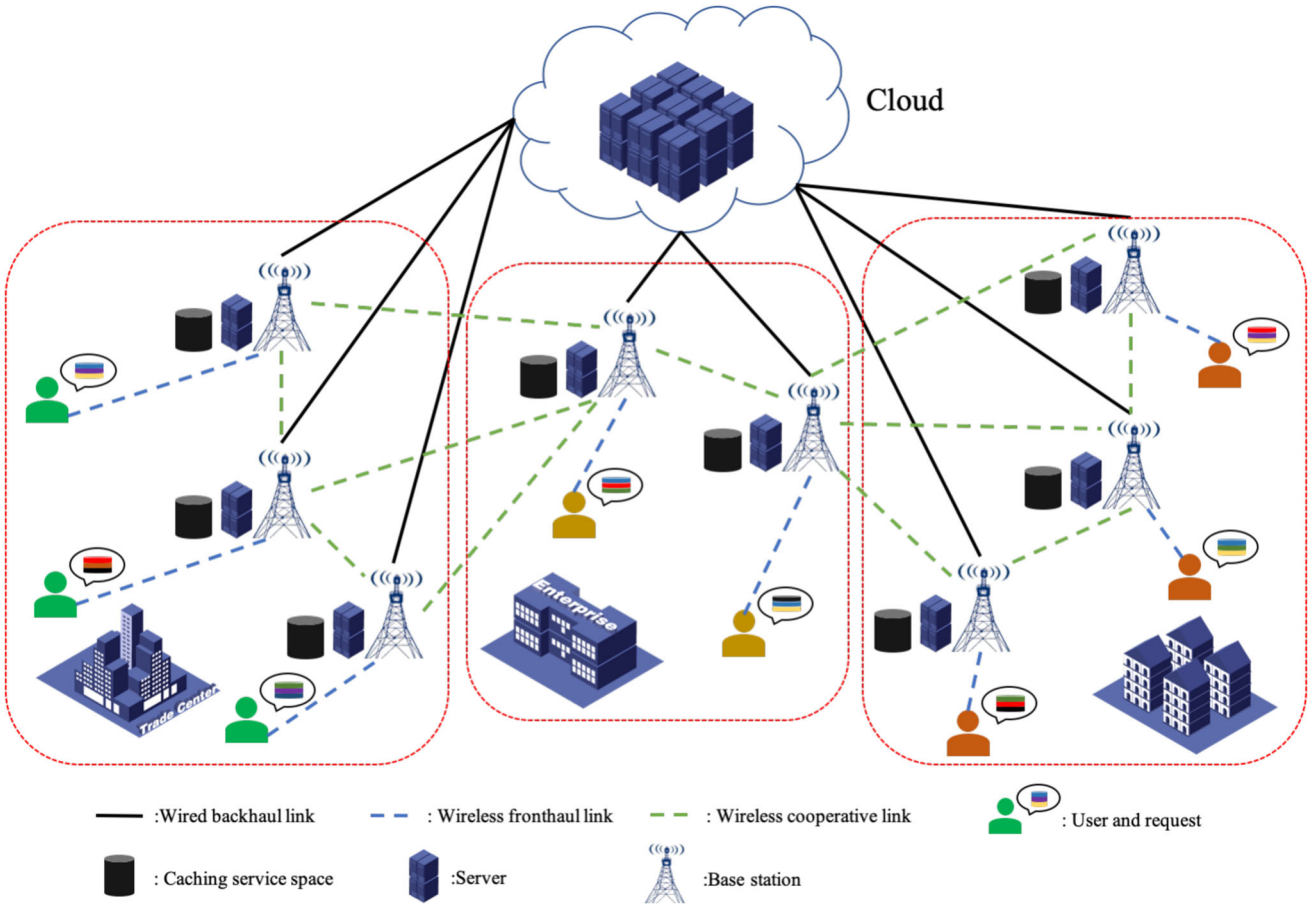

Fig. 1 Cloud-edge collaborative multi-scenario service caching system architecture

In the cloud-edge computing system, a total of E edge servers, denoted as $\mathcal{E} = \{1, 2, \ldots, E\}$, are connected to the cloud server. Each edge server communicates with other edge servers and the central cloud through a base station. The storage capacity and computing resources of an edge server $e \in \mathcal{E}$ are denoted by G_e and F_e, respectively. A total of H services, denote as $\mathcal{H} = \{1, 2, \ldots, H\}$, can be requested by users. The storage size occupied by application service h is denoted as $\varrho_h \in \mathbb{R}^+, \forall h \in \mathcal{H}$. Normally, we assume that all application services in $\mathcal{H}$ are stored on the central cloud C. The application services stored on edge server e are denoted by $\mathcal{H}_e$. Due to limited storage capacity, the services on each edge server must meet the constraint $\sum_{h \in \mathcal{H}_e} g_h \leq G_e$. The users of the edge server service are represented as $\mathcal{U} = \{1, 2, \ldots, U\}$, randomly distributed across regions served by edge servers. Each user u can request any application service in $\mathcal{H}$. Let q_u represent the request sent by user u to an edge server through the access link. Each user request q_u includes the application service h, the required computing resources X_h, and the data size of request D_h. The user request is defined as $q_u = \{h, X_h, D_h\}$. We assume that the total time $\mathcal{T}$ consists of T time slots, denoted as $\mathcal{T} = \{1, 2, \ldots, T\}$. Notably, we assume that the operation of each service caching is completed within a single time slot. Additionally, each user can request only one service per time slot, and the request must be processed within the same time slot. We assume each user is served by a single adjacent edge server, and this association remains fixed within each time slot.

If a requested service is not cached on the edge server, the arriving requests cannot be processed. Therefore, the edge node must carefully decide which services to cache and how much computational resource to allocate to heterogeneous tasks in an online manner.

Specifically, The service caching decision for the edge server is denoted as $\psi_{h,e}^t \in \{0, 1\}$, which is a binary variable and the computational resource allocation decision is denoted as $f_{h,e}^t$, which is a continuous variable. It should be emphasized that $\psi_{h,e}^t = 1$ indicates service h is cached on the edge server e, while $\psi_{h,e}^t = 0$ means it is not. Clearly, if $\psi_{h,e}^t = 0$, then $f_{h,e}^t = 0$, meaning no computational resource are allocated to service h if it is not cached on the edge server e. Therefore, the entire decision for service caching on edge server e in time slot t is represented as $Y_e^t = \{\psi_e^{1,t}, \psi_e^{2,t}, \ldots, \psi_e^{H,t}\}$ for service placement

and $\mathcal{F}_e^t = \{f_e^{1,t}, f_e^{2,t}, \ldots, f_e^{H,t}\}$ for resource allocation.. In addition, the maximum available computational resource for edge server e is represented by f_e^{max}. Therefore, there exist

$$\sum_{h=1}^{H} f_e^{h,t} \leqslant F_e, \forall t, e \tag{1}$$

3.2 Latency model

The following sections describe the latencies involved in processing requests within the edge computing system, specifically: serving latency, and transmission latency.

Based on the network model, user requests is processed on local edge server, a neighboring edge server, or the central cloud. If $\psi_e^h = 0$ and $\sum_{j \in e_N} \psi_j^h >= 1$, service h is cached on neighborhood of the edge server e. If $\sum_{j \in \{J_e, e\}} \psi_j^h = 0$, service h is not cached on the edge server e and its neighbour J_e. Next, we develop the user request latency model in three situations.

When $\psi_e^h = 1$, the local edge server e has cached the requested service h and will process user request r_u locally. In this system, the total latency includes request transmission latency and data calculation latency. In this case, the total latency of user request r_u can be calculated by:

$$T_{u,e}^h = \frac{X_u^h}{f_e^h} + \frac{D_u^h}{M_{u,e}} \tag{2}$$

where X_u^h is the computing resources required to complete the service requested by user, while f_e^h represents the unit computing resources allocated by edge servers e to service h. The term $\frac{X_u^h}{f_e^h}$ indicates the computing latency of request r_u handled by the local edge server e . In the second item, D_u^h represents the input data size for the request r_u, $M_{u,e}$ denotes uplink transmission rate of user u and $\frac{D_u^h}{M_{u,e}}$ indicates the transmission time that the service request is uploaded to the edge server e.

When $\psi_e^h = 0$ and $\sum_{j \in J_e} \psi_j^h >= 1$, the local edge server e has not cached the relevant service h for user request r_u, but its neighbour J_e has cached the service h and decides to forward the user requests r_u to J_e for processing. Therefore, the request transmission latency should include the uplink transmission time from user u to local edge server e, as well as the time from edge server e to neighbor server $j \in J_e$. In this case, the total latency for user request r_u can be calculated as follow:

$$T_{u,j}^h = \frac{X_u^h}{f_j^h} + \frac{D_u^h}{M_{u,e}} + \frac{D_u^h}{M_{e,j}} \tag{3}$$

where f_j^h denotes the unit computing resources allocated by the adjacent server j to service h and $\frac{X_u^h}{f_j^h}$ represents the computing latency for request r_u handled by the adjacent server j. The second item, $\frac{D_u^h}{M_{u,e}}$, represents the transmission time that the service request is uploaded to the local edge server e. Additionally, $\frac{D_u^h}{M_{e,j}}$ represents the uplink transmission time from local server e to the adjacent server j, where $M_{e,j}$ denotes the upload bandwidth from edge server e to adjacent server j. In our system, we assume that $M_{e,j}$ are constant values.

When $\sum_{j \in \{J_e, e\}} \psi_j^h = 0$, the local edge server e has not cached the relevant service h requested by the user, and its adjacent servers J_e also has not cached the service h. The user requests r_u must be sent to the central cloud c. Thus, the request transmission latency includes the transmission time that the service request is uploaded to the local edge server e, as well as the transmission time from edge server e to the central cloud c. In this scenario, the total latency for user request r_u can be calculated by:

$$T_{u,c}^h = \frac{X_u^h}{f_c^h} + \frac{D_u^h}{M_{u,e}} + \frac{D_u^h}{M_{e,c}} \tag{4}$$

where f_c^h denotes the unit computing resources allocated by the central cloud c to service h, while $\frac{X_u^h}{f_c^h}$ represents the computing latency for request r_u handled by the central cloud c. The third component, $\frac{D_u^h}{M_{e,c}}$ represents the uplink transmission time from the local edge server e to the central cloud C, and $M_{e,c}$ is the bandwidth uploaded from edge server e to central cloud c. In our system, we assume that $M_{e,c}$ are constant values.

3.3 Problem formulation

Considering that user requests arrive at the edge server randomly, we aim to develop an optimal service caching strategy for each edge server with the goal of minimizing the response time of all user requests. The problem is formally formulated as follows:

$$\min \sum_{t}^{T} \sum_{e}^{E} \sum_{u}^{U} \sum_{h}^{H} p_e^{h,t} (\psi_e^h \times T_{u,e}^{h,t} + (1 - \psi_e^{h,t}) \times \min(1, \sum_{j}^{J_e} \psi_j^{h,t}) \times T_{u,j}^{h,t} + (1 - \psi_e^{h,t}) \times (1 - \min(1, \sum_{j}^{J_e} \psi_j^{h,t})) \times T_{u,c}^{h,t}) \tag{5a}$$

$$s.t.\ \psi_e^{h,t} \in \{0, 1\}, \quad \forall e \in \mathcal{E}, h \in \mathcal{H}, t \in \mathcal{T} \tag{5b}$$

$$\sum_{h}^{\mathcal{H}_e} g_e^h \leqslant G_e, \quad \forall e \in \mathcal{E} \tag{5c}$$

$$\sum_{h}^{\mathcal{H}_e} f_e^{h,t} \leqslant f_e^{max}, \quad \forall e \in \mathcal{E}, t \in \mathcal{T} \tag{5d}$$

$$0 \leqslant f_e^{h,t} \leqslant f_e^{max}, \quad \forall e \in \mathcal{E}, h \in \mathcal{H}, t \in \mathcal{T} \tag{5e}$$

Constraint Eq. 5b specifies each edge server has two options for caching service. Constraint Eq. 5c indicates that the total capacity of all the services cached on the edge server must not exceed the edge server's storage capacity. Constraints Eqs. 5d and 5e state that the total computing resources allocated to all services by the edge server must be within its computing capacity, and that each cached service must receive a positive allocation of resources. According to Eqs. 5b and 5e, our objective function Eq. 5a involves both continuous variables and binary variables, which obviously is a mixed integer nonlinear programming problem.

The multi-commodity facility location problem [33] The multi-commodity facility location problem is well-known to be NP-hard. In this problem, each user requests different commodities, and the cost of obtaining these commodities varies depending on the facility location. The objective is to select l locations from a set of candidate sites S to establish facilities and determine which commodities to produce at each location, in order to minimize the total cost for users.

In the edge service caching problem, we consider E edge servers and F distinct application services. Each user u requests different services from the edge servers, and the response time for each service varies. We can model the edge servers as facilities and the services as commodities. Thus, the edge service caching problem can be formulated as an instance of the multi-commodity facility location problem. Moreover, in this problem, the edge servers must also allocate resources to the deployed services, which further complicates the problem. Therefore, the edge service caching problem is also NP-hard. Traditional optimization algorithms struggle to solve this problem within polynomial time. DRL provides a feasible and effective solution for problem.

4 Algorithm design

In this section, we first model the service placement and resource allocation problem for multi-edge servers as a decentralized Markov decision process (DMDP). Next, we introduce a cooperative multi-agent based approach.

4.1 Reformulated problem

A typical DMCP is defined as a tuple with six elements, $\langle I, S, A, P, R, \gamma \rangle$. Specifically, I represents a set of agents and S represents a set of joint states. The joint action space is represented as $A = A_1 \times A_2 \times \ldots \times A_E$, where $a = (a_1, a_2, \ldots, a_E)$ denotes the joint actions, and A_e is the action space for agent e. The transition probability function $P : S \times A \times S \rightarrow [0, 1]$ denotes the probability of transition from state $s \in S$ to $s' \in S$ under a given action $a \in A$. $R : S \times A \rightarrow R$ represents the reward function and $\gamma \in [0, 1]$ represent a discount factor indicating the weight of future rewards. As follow, the cooperative service caching problem is formulated by the MARL framework.s

Agent $\mathcal{E} = \{1, 2, \ldots, E\}$ refers a set of edge servers. The storage resources and computing resources of each edge server are denoted by G_e and F_e, respectively.

State The state space of cloud-edge collaborative service caching problem is represented as $\mathcal{S} = \{S^1, S^2, \ldots, S^T\}$, where each S^t represents joint state of all the information observed from the edge computing system at time t, i.e. $S^t = (s_1^t, s_2^t, \ldots, s_E^t)$. s_e^t represent the state of agent e observed from the system environment at time slot t. And s_e^t is denoted as $s_e^t = \{\mathcal{H}_e^t, \mathcal{G}_e^t, \mathcal{F}_e^t, \mathcal{R}_e^t, \mathcal{J}_e\}$, where $\mathcal{H}_e^t = \{h_e^t\}$ indicates service h is cached on the edge server e in time slot t. $\mathcal{G}_e^t = \{g_e^{h,t}\}, \forall h \in \mathcal{H}_e$ represents the storage occupied by cached services $\mathcal{H}_e$ on edge server e in time slot t, $\mathcal{F}_e^t = \{f_e^{h,t}\}, \forall h \in \mathcal{H}_e$ denotes the computing resources allocated by edge server e to cached services h in time slot t. $\mathcal{R}_e^t$ represents the set of requests received by edge server e in time slot t and $\mathcal{J}_e$ denotes the adjacent servers of edge server e.

$$O_e^t = \{S_e^t, \{S_j^t\}_{j \in J_e}, P_e^t\} \tag{6}$$

where S_e^t represent the state of agent e, S_j^t is the state of adjacent server j and $P_e^t = \{p_e^{h,t}\}, \forall h \in \mathcal{H}_e$, $p_e^{h,t} \in [0, 1]$ is the probability that a user requests the service h at edge server e in time slot t.

Actions $\mathcal{A}^t = \{A^1, A^2, \ldots, A^T\}$ where A^t is the joint action of all the agent at each time slot t, specifically, $A^t = (a_1^t, a_2^t, \ldots, a_E^t)$. The action of edge server e at time slot t is denoted as $a_e^t = \{Y_e^t, F_e^t\}$, where $Y_e^t = \{\psi_e^{1,t}, \psi_e^{2,t}, \ldots, \psi_e^{H,t}\}, \forall h \in \mathcal{H}, \psi_e^{t,h} \in \{0, 1\}$ and $F_e^t = \{f_e^{1,t}, f_e^{2,t}, \ldots, f_e^{H,t}\}, \forall h \in \mathcal{H}, 0 \leqslant f_e^{h,t} \leqslant f_e^{max}$. If edge server e decides to cache service h and allocate computing resources $f_e^{h,t}$ to it in time slot t, then $\psi_e^{h,t} = 1$, otherwise, $\psi_e^{h,t} = 0$.

Reward Function At time slot t, after every agent e takes action a_e^t, the entire system's state changes from S^t to S^{t+1}, and all the edge servers receives rewards. We define the total reward for all the edge server at time slot t as the negative total latency incurred by all edge servers in responding to user

requests. The reward function for all edge servers is defined as follows:

$$r^t = -\sum_{e}^{E}\sum_{u}^{U}\sum_{h}^{H} p_e^{h,t}(\psi_e^h \times T_{u,e}^{h,t} + (1-\psi_e^{h,t}) \times \min(1, \sum_{j}^{J_e} \psi_j^{h,t}) \times T_{u,j}^{h,t} + (1-\psi_e^{h,t}) \times (1-\min(1, \sum_{j}^{J_e} \psi_j^{h,t})) \times T_{u,c}^{h,t}) \tag{7}$$

4.2 Algorithm design

Since the service caching and resource allocation problem involves both continuous and discrete action spaces, we devise a DRL algorithm based on the soft actor-critic (SAC) framework with a hybrid discrete-continuous action space. SAC is an offline reinforcement learning algorithm based on policy gradients. The Multi-agent SAC is an extension of the Soft Actor-Critic (SAC) algorithm, specifically designed for multi-agent scenarios. It combines SAC's entropy maximization strategy with the cooperation mechanisms inherent in multi-agent systems, enabling multiple agents to learn effective strategies in complex environments.

We use SAC to develop an optimal strategy that minimizes expected delay. In order to increase the randomness and exploration of the strategy, SAC adds entropy item $\mathcal{H}(\pi(\cdot|s^t))$ to the learning objective. The soft Q-value function is defined as follows:

$$Q(s^t, a^t) = \sum_{t=0}^{T} \mathbb{E}[\gamma^t r^t(s^t, a^t) + \alpha\gamma^{t+1}\mathcal{H}\sum_{e\in\mathcal{E}}(\pi_e(\cdot|s_e^t))], \tag{8}$$

where γ is the discount factor and α is the temperature parameter.

Within each time slot, the edge server e first observes its own state and then gets the current service cache action through the policy network. After the edge server caches the services according to its actions, the user's service request is processed by the local edge server. If the requested service is not cached, the request is forwarded to either an adjacent server or the cloud server. In each time slot, the total delay incurred by each server in responding to all requests serves as the system's reward. Consequently, the system maintains a tuple denoted as (S^t, A^t, r^t, S^{t+1}), which includes the joint status of all edge servers at time slot t, the joint action, the total reward, and the joint status for the next time slot $t+1$. The type of service requested by users varies in each time slot, leading the edge server to make different service caching decisions accordingly.

We represent actor network (policy network) as $\pi_{\omega_e}(a_e^t|s_e^t)$ and the critic network (value network) as $Q_\theta(s^t, a^t)$. The detailed neural network structure is illustrated in Fig. 2. The model based on known service popularity fails to capture the time-varying nature of user service requests and their correlations. Additionally, incorporating all historical requests into the observation significantly expands the state space, complicating the training process. To address this issue, we employ a Transformer network to predict the distribution of user service requests. This model, based on the attention mechanism, efficiently processes sequence data. The Transformer architecture consists of multiple layers of encoders and decoders. The encoder extracts information from the input sequence, while the decoder generates the output sequence. Specifically, prior to predicting the requested service, we first encode each service request. As neural networks cannot directly process raw categorical data, all input and output variables are converted into numerical formats for compatibility with the model. Once the data is encoded, the self-attention mechanism allows the Transformer to focus on service requests

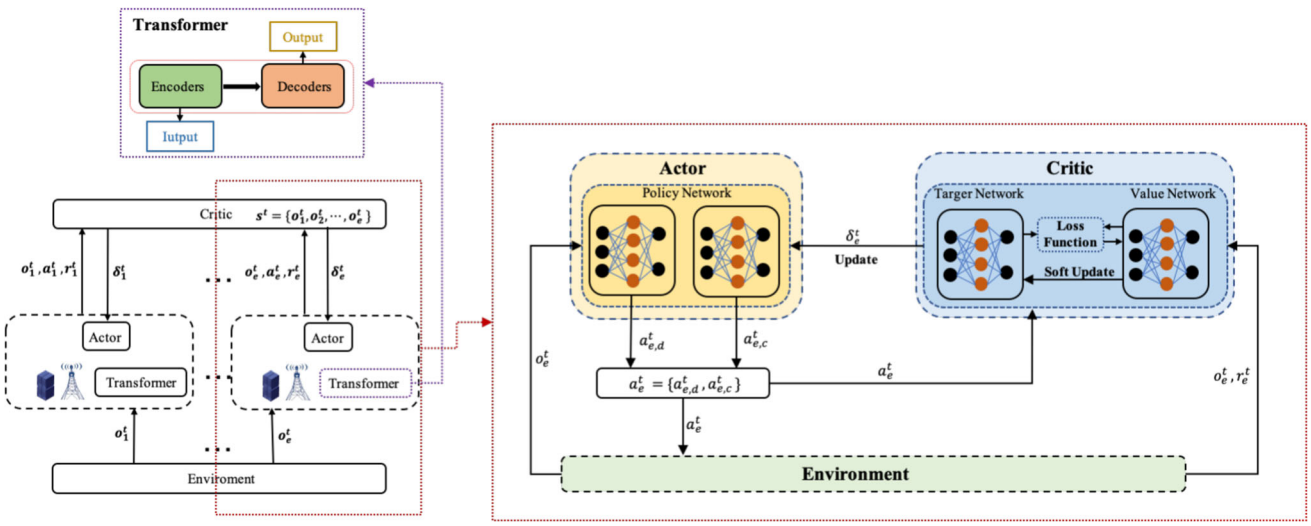

Fig. 2 Service caching framework with multi-agent soft actor-critic reinforcement learning algorithms

across all time steps when processing the current request, thereby capturing the global dependencies of user service requests. Ultimately, the attention vector is utilized to predict the service request. Intuitively, the requested service can be viewed as a category, with the decoder ultimately outputting the probability of service h requested by the user, denoted as p_e^h.

Critic. We design a value networks as the critic which is denoted as $Q_\theta(s^t, a^t)$ with parameter θ and a target networks which is denoted as $Q_{\hat{\theta}}(s^t, a^t)$ with parameters $\hat{\theta}$ to reduce the overestimation of state values by the value network. We separately update θ and $\hat{\theta}$. The value network $Q_\theta(s^t, a^t)$ is trained by minimizing the loss function, defined as follows:

$$L(Q_\theta) = \mathbb{E}_{(s^t,a^t)\sim\mathcal{M}}[\frac{1}{2}(Q_\theta(s^t, a^t) - y^t)^2] \tag{9}$$

where y_e^t is the estimation of the value function based on the actual observed reward r^t, which is defined as:

$$y^t = r(s^t, a^t) + \gamma \mathbb{E}_{(s^t,a^t)\sim\rho_\pi}[Q_\theta(s^{t+1}, a^{t+1}) - \alpha \sum_{e\in\mathcal{E}} log\pi_{\omega_e}(a_e^{t+1}|s_e^{t+1})] \tag{10}$$

Then the parameters of value network θ is updated by $\Delta L(Q_\theta)$. In order to ensure the stability of model training, the parameters of target networks $\hat{\theta}$ is updated by θ , which is computed as:

$$\hat{\theta} = \tau\theta + (1 - \tau)\hat{\theta} \tag{11}$$

where $\tau \in (0, 1)$ is the updata factor.

Actor. We design a policy network as the actor for agent e which is denoted as $\pi_{\omega_e}(a_e^t|s_e^t)$. The parameters of policy network ω_e can be learned by minimizing the following formula:

$$J(\omega_e) = \mathbb{E}_{s_e^t\sim\mathcal{M}}[D_{KL}(\pi_{\omega_e}(\cdot|s_e^t)\| \frac{exp(Q_\theta(s_e^t|\cdot))}{Z_\theta(s_e^t)})] \tag{12}$$

where $D_KL(p\|q)$ is the Kullback-Leibler (KL) divergence that evaluate the difference between p and q. Since $Z_\theta(s_e^t)$ has no effect on strategy updates, we can rewrite the Eq. 12 as:

$$J(\omega_e) = \mathbb{E}_{s_e^t\sim\mathcal{M}}\mathbb{E}_{a_e^t\sim\pi_{\omega_e}}[\sum_{e\in\mathcal{E}} \log \pi_{\omega_e}(a_e^t|s_e^t) - Q_\theta(s_e^t, a_e^t)] \tag{13}$$

The detailed training process for the service cache algorithm is presented in Algorithm 1. First, we initialize the system state and the parameters of the critic and actor networks (line 1-3). Each agent e observes its local state at the start of each time slot. It then determines its actions by inputting observations into the local actor network (line 6). After all agents have determined their actions, they record their local state for the next time slot (line 7). The central controller aggregates the states observed by each edge server for both the current and next time slots. Following the local actions of all edge servers, the overall server response time is calculated based on user service requests to derive the reward (line 9). The central controller inputs the state and actions of all servers in the current time slot into the value network to obtain the state-action prediction value (line 10). Simultaneously, the state and actions of all servers in the next time slot are input into the value network to obtain the next time slot's prediction value (line 11), followed by error calculation. The value network is updated by minimizing the calculated error (line 13), followed by the update of the target network (line 14). Finally, each edge server updates its local actor network based on the calculated error (lines 15-16).

Algorithm 1 MATHSAC Algorithm.

1: Initialize each edge server's actor networks with parameters ω_e. Initialize the critic networks with parameters θ and $\hat{\theta}$.
2: **for** each episode = 1 : N **do**
3: Reset the environment.
4: **for** time step $t = 1 : T$ **do**
5: **for** each $e \in \mathcal{E}$ **do**
6: Get observation O_e^t and Select an action $a_e^t \leftarrow \pi_{\omega^e}(\cdot|O_e^t)$.
7: Each edge server execute the action and obtain the next observation O_e^{t+1}.
8: **end for**
9: Get the observation of all edge server $s^t = \{O_1^t, O_2^t, \ldots, O_E^t\}$, the next observation $s^{t+1} = \{O_1^{t+1}, O_2^{t+1}, \ldots, O_E^{t+1}\}$ and the reward r^t.
10: Value network makes predictions: $Q_\theta(s^t, a^t)$
11: Target network makes predictions: $Q_{\theta'}(s^{t+1}, a^{t+1})$
12: $y^t = r^t + \gamma Q_{\theta'}(s^{t+1}, a^{t+1})$
13: Update the value network according to minimizing Eq. 9
14: Update the target network according to Eq. 11
15: **for** each edge server $e \in \mathcal{E}$ **do**
16: Update the policy network according to Eq. 13
17: **end for**
18: **end for**
19: **end for**
20: Return result

The computational complexity of the algorithm primarily depends on the parameter calculations of the actor and critic networks, which include convolutional and fully connected networks. A convolutional networks's computational complexity is represented by $C_c = \mathcal{O}(\sum_l^L d_m^2 d_k^2 I_l I_{l-1})$, where L is the number of convolutional layers, d_m is the feature map size, d_k as the kernel size, and I is the number of filters. A fully-connected networks's computational complexity is denoted by $C_f = \mathcal{O}(\sum_m^M u_m u_{m-1})$, where u_m is the number of neural units in fully-connected layer m. Thus, the computational complexity of actor and critic is $\mathcal{O}(C_c + C_f)$. The critic in our algorithm consists of a value network and a target network, while each agent's actor comprises two action

networks. Therefore, the total computational complexity of Algorithm 1 is $\mathcal{O}(2NT(E+1)(C_c+C_f))$.

5 Performance evaluation

5.1 Simulation setup

In this chapter, we evaluate the performance of our multi-agent based on transformer hybrid soft actor-critic algorithm (MATHSAC), using simulated data. We present the experimental setup and implementation details as follows.

This article presents an algorithm designed to address the problem of service deployment in edge computing scenarios. According to [31, 34], We simulated an edge computing scenario that included 12 edge servers and 30 services. We assume that the caching capacity of each edge server is randomly selected from the range of 1 to 3 GB, while the computing frequency of the server's central processing unit (CPU), representing its computing power, is randomly selected from the range of 10 to 30 GHz. We assume that the storage resources utilized by the service model follow a uniform distribution ranging from 100 to 500 MB, while the computational resources required for each service follow a uniform distribution ranging from 100 to 200 GB.

The experiments are conducted using Python 3.10 on an Ubuntu 64-bit system with a 2.3 GHz, 12-core Intel Xeon Gold 5118 processor and 64 GB of memory. We utilize PyTorch to implement the actor and critic neural networks, which consist of an input layer, four hidden layers with 256, 512, 256, and 64 nodes, and an output layer. The input and output layer sizes depend on the dimensions of the state-action space. In the MATHSAC settings, each edge server executes the policy and interacts with the environment 500 times, storing the state, action, and reward for each interaction. In the simulation environment, the neighbor count for each edge server is set to between 1 and 3 to simplify the regional distribution. The learning rate is set to $\eta = 0.001$ and the discount factor to $\gamma = 0.94$.

- *PGDDQN* : a novel DDQN-based method with generative adversarial networks (GAN) and prioritized experience replay (PER) [31].
- *MAPPO* : Multi-Agent Proximal Policy Optimization is an extension of PPO (Proximal Policy Optimization) [35].
- *DDQN* : Each agent uses two Q-networks to serve the training server policy, thereby eliminating the problem of overestimation [36].
- *Random* : Each edge server caches application services randomly, limited by its maximum storage capacity.

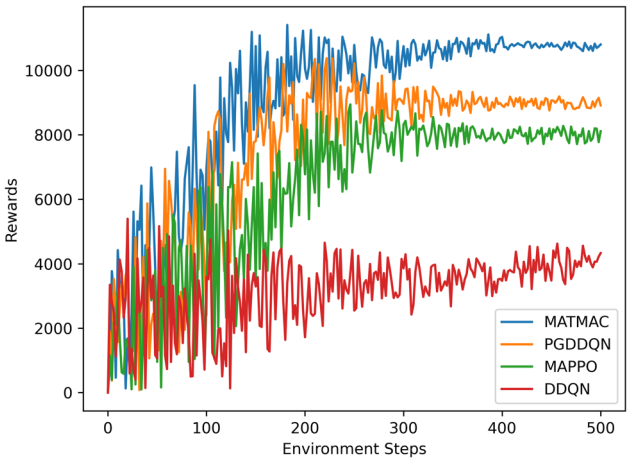

Fig. 3 Convergence Performance of MATHSAC and the three baselines

We employ total service response latency and service hit rate as metrics for performance evaluation.

5.2 Evaluation results

First, we compare the training performance of our algorithm with two other reinforcement learning methods. Figure 3 illustrates the total rewards of MATHSAC, PGDDQN, MAPPO and DDQN algorithms in a simulated environment. DDQN lacks robustness and fails to adapt to all agents, resulting in the lowest total reward. Compared to PGDDQN, MAPPO, MATHSAC leverages a policy network to more accurately evaluate environmental state values, achieving a higher total reward. Specifically, MATHSAC's total reward stabilizes around 11,000 after 400 environmental steps, whereas MADRL's total reward stabilizes around 9,000 after 350 steps. MATHSAC demonstrates a 18.5% improvement over MADRL. We observe that MATHSAC stabilizes after

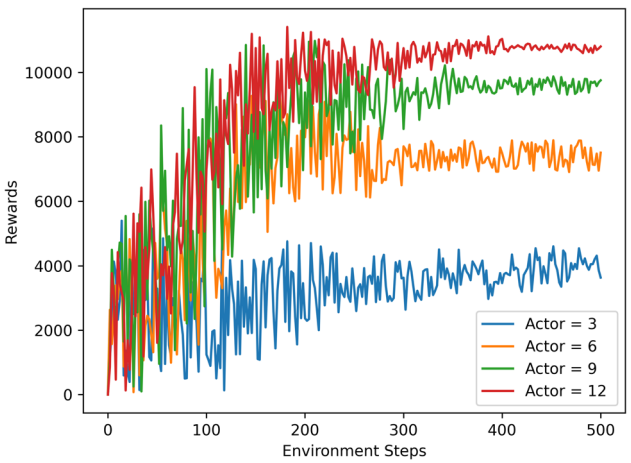

Fig. 4 Performance of the MATHSAC's training process with varying numbers of agents

400 steps, converging more slowly than MADRL but with reduced fluctuations. This is due to MATHSAC's inclusion of a maximum entropy term, which slows convergence but improves stability, thereby reducing fluctuations.

Figure 4 explores the impact of the number of actors on the training performance of the proposed algorithm. As shown in Fig. 4, the reward value increases consistently with the number of actors. When the number of steps reaches 500, the reward value increases to 7500, reflecting a 80% improvement compared to training with 3 actors. As more actors collaborate within the value network, the action space expands, and the value network's predictions become more accurate. Additionally, the reward curves for 12 actors are more stable than those for 9 actors, further demonstrating the effectiveness and robustness of the proposed algorithm.

Figure 5 illustrates the performance of the four algorithms across varying numbers of edge servers. The number of our edge servers varies from 9 to 45. Overall, Fig. 5(a) shows that the three reinforcement learning-based approaches for service caching outperform the randomized service caching algorithm in terms of total response latency, across systems with varying numbers of edge servers. This is due to the fact that reinforcement learning based algorithms have the learning ability to generate more informed service caching actions. Notably, MATHSAC achieves the lowest total delay, outperforming the others. As the number of edge servers increases, the total response latency for all four algorithms rises, since the system incurs higher costs to handle more caching requests. After assessing the algorithms' total delay performance, we evaluated their local edge server hit rates, as shown in Fig. 5(b). The edge hit rates follow this ranking: MATHSAC > PGDDQN > MAPPO > Random. The random policy cannot adjust the edge server's cache services according to user requests. As a result, the random policy has the lowest local server hit rate. Unlike the other algorithms, MATHSAC considers interactions between local and neighboring servers and predicts user request types, allowing it to obtain the largest hit rate across varying numbers of edge servers.

The number of service types refers to the variety of services a user can request. As shown in Fig. 6(a), the total service delay increases as the number of service types grows, since a larger number of services expands the agent's action space. However, limited storage capacity means user requests cannot always be fulfilled locally or by neighboring servers, and may require retrieval from the cloud, leading to higher overall delay. Similarly, Fig. 6(b) shows that the local hit rate decreases as the number of service types increases.

Figure 7(a) compares the total response service delay of MATHSAC with three other algorithms across varying edge server storage capacities. Under different server storage sizes, MATHSAC consistently outperforms the other three algorithms in terms of total response service delay. As the storage capacity of edge servers increases, the total response service delay for all algorithms decreases. This is because larger storage allows more services to be cached, increasing the likelihood that user requests can be served locally or by neighboring edge servers, reducing the overall delay. We note that the total delay of our algorithm is approximately 35% lower than the random caching strategy. Figure 7(b) illustrates the effect of edge server storage capacity on local server hit rate. As edge server storage increases, the hit rate of all algorithms improves. In addition, the local server hit rate under MATHSAC rises more rapidly compared to the other two RL-based algorithms. However, as edge server storage continues to grow, the rate of increase in MATHSAC's hit rate slows down. This occurs because, once server storage reaches a certain threshold, all services are fully cached, and the hit rate plateaus.

The data size of a user request refers to the amount of data the user sends to the server for service processing. Figure 8(a) illustrates how the size of user request data affects total response service delay across four service caching methods. Under varying user request data sizes, the total delay of the three reinforcement learning-based caching methods is consistently lower than the random caching algorithm. Notably, MATHSAC achieves the lowest total delay. As user request data size grows, total response service delay for all four algorithms increases due to longer transmission and processing times. Similarly, Fig. 8(b) shows a decrease in local hit rate as user request data size increases. Notably, as user request data size grows, The hit rate of MATHSAC decreases

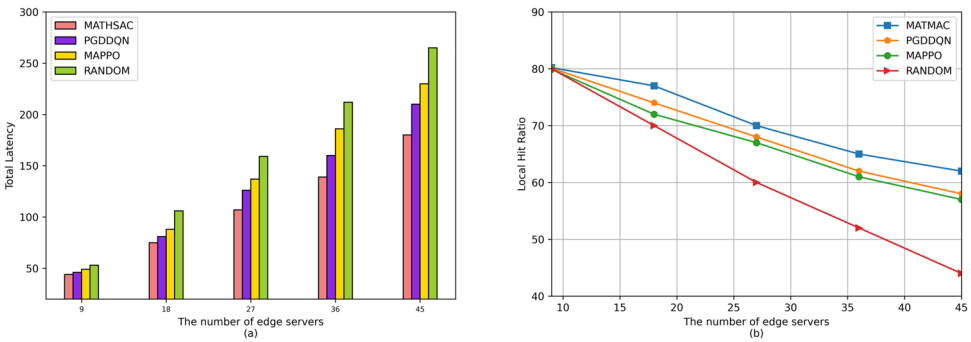

Fig. 5 Performance of MATHSAC and the three baselines with different number of edge servers

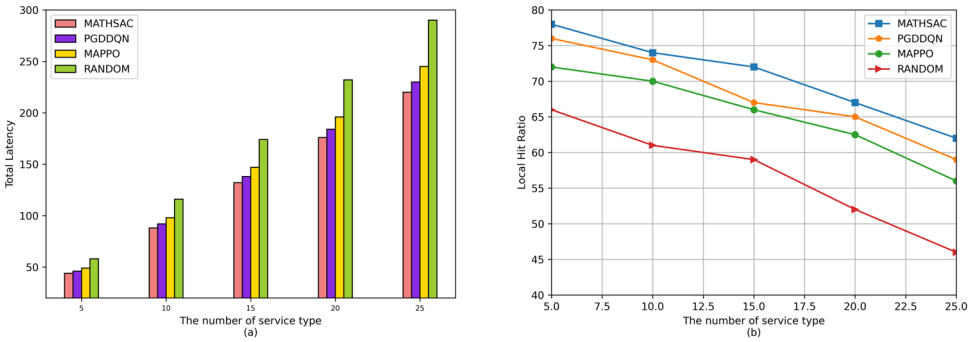

Fig. 6 Performance of MATHSAC and the three baselines with different number of service type

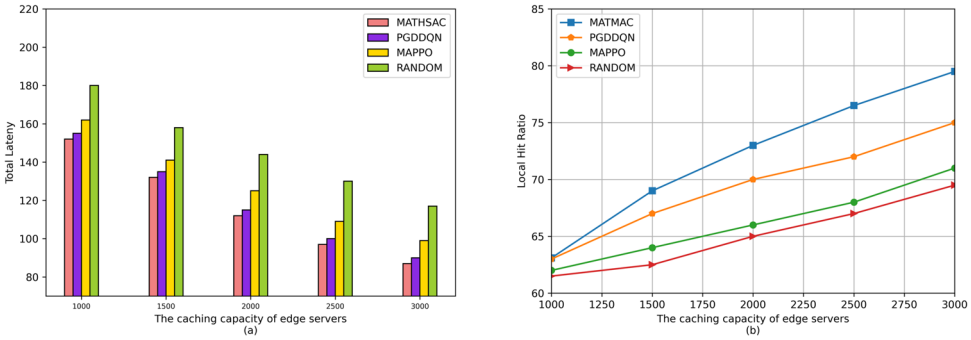

Fig. 7 Performance of MATHSAC and the three baselines with different caching capacity of edge servers

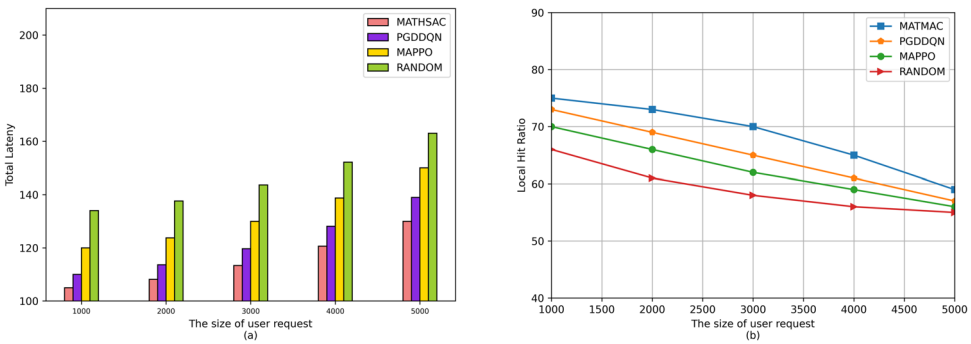

Fig. 8 Performance of MATHSAC and the three baselines with different sizes of user request

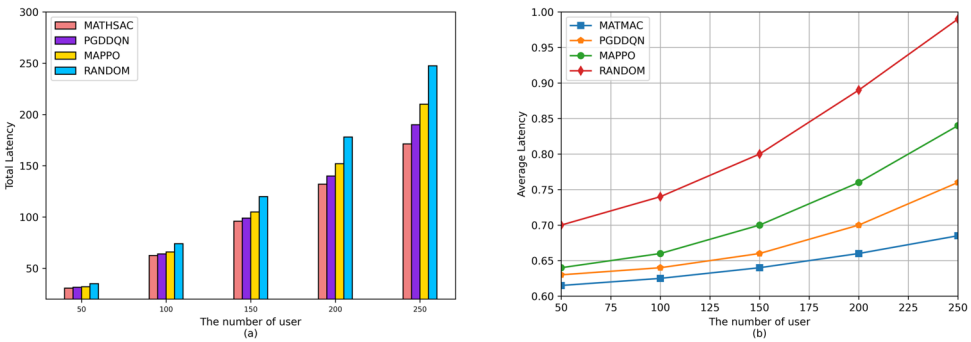

Fig. 9 Performance of MATHSAC and the three baselines with different number of users

more rapidly. This is because server's storage and computing power limitations result in more requests being forwarded to the cloud server for response.

To evaluate the algorithm's performance in large-scale scenarios, we tested it under varying numbers of users. As shown in Fig. 9, the average response delay increases with the growing number of users and expanding network scale across all schemes. The three reinforcement learning algorithms outperform the random placement approach. This is because the random scheme lacks the ability to dynamically adjust the service caching strategy as the user count increases. In contrast, the reinforcement learning algorithms can continuously adapt their strategies as the number of users grows, resulting in greater stability. However, the proposed MATHSAC scheme consistently outperforms the others by predicting user preferences, perceiving environmental changes in real time, and efficiently allocating computing resources to application services.

6 Conclusion

This paper addresses the issue of cloud-edge collaborative service caching, aiming to minimize service response latency under limited storage resources. Considering the dynamic and time-varying nature of user requests, we propose a transformer-based method. Each agent utilizes a transformer network to capture the correlations in user service requests, improving the prediction of the requested service type. To solve service caching problem in heterogeneous edge networks, we introduce a multi-agent hybrid action soft actor-critic algorithm. In this approach, all agents share a critic network to learn the heterogeneity of service requests. Each agent observes the state of neighboring agents to enhance collaboration, while simultaneously training two policy networks: one to decide which service to cache, and the other to allocate computing resources. Experimental results, based on multiple evaluation metrics, demonstrate that our proposed caching strategy outperforms other methods. The emergence of large language model services will complicate service deployment. These services typically consist of numerous interdependent basic models, which are tightly coupled. Deploying such services requires careful consideration of the dependencies and interactions among the basic models. Additionally, as network scale grows, the algorithm's stability and scalability must be reassessed. These challenges will be addressed in future work.

Author Contributions Y.L. wrote the main manuscript text, conducted simulation experiments and prepared all the figures. Z.Z. provided theoretical guidance. H.C. revised and polished the article. All authors reviewed the manuscript.

Funding This work has been funded by the National Natural Science Foundation of China under grant 62173026.

Data Availability No datasets were generated or analysed during the current study.

Declarations

Competing Interests The authors declare no competing interests.

References

1. Iftikhar S, Gill SS, Song C, Xu M, Aslanpour MS, Toosi AN, Du J, Wu H, Ghosh S, Chowdhury D, et al. (2023) Ai-based fog and edge computing: A systematic review, taxonomy and future directions. Internet of Things 21:100674
2. Barrios C, Kumar M (2023) Service caching and computation reuse strategies at the edge: A survey. ACM Comput Surv 56(2):1–38
3. Chai F, Zhang Q, Yao H, Xin X, Gao R, Guizani M (2023) Joint multi-task offloading and resource allocation for mobile edge computing systems in satellite iot. IEEE Trans Veh Technol 72(6):7783–7795
4. Qiu T, Chi J, Zhou X, Ning Z, Atiquzzaman M, Wu DO (2020) Edge computing in industrial internet of things: Architecture, advances and challenges. IEEE Commun Surv Tutor 22(4):2462–2488
5. Sharma M, Tomar A, Hazra A (2024) Edge computing for industry 5.0: fundamental, applications and research challenges. IEEE Internet of Things J
6. Apat HK, Nayak R, Sahoo B (2023) A comprehensive review on internet of things application placement in fog computing environment. Internet of Things, 100866
7. Ju Y, Chen Y, Cao Z, Liu L, Pei Q, Xiao M, Ota K, Dong M, Leung VC (2023) Joint secure offloading and resource allocation for vehicular edge computing network: A multi-agent deep reinforcement learning approach. IEEE Trans Intell Transport Syst 24(5):5555–5569
8. Gao H, Wang X, Wei W, Al-Dulaimi A, Xu Y (2023) Com-ddpg: Task offloading based on multiagent reinforcement learning for information-communication-enhanced mobile edge computing in the internet of vehicles. IEEE Trans Veh Technol
9. Kim M, Lee H, Hwang S, Debbah M, Lee I (2024) Cooperative multi-agent deep reinforcement learning methods for uav-aided mobile edge computing networks. IEEE Internet of Things J
10. Tilahun FD, Abebe AT, Kang CG (2023) Multi-agent reinforcement learning for distributed resource allocation in cell-free massive mimo-enabled mobile edge computing network. IEEE Trans Veh Technol
11. Bi S, Huang L, Zhang Y-JA (2020) Joint optimization of service caching placement and computation offloading in mobile edge computing systems. IEEE Trans Wireless Commun 19(7):4947–4963
12. Zhou H, Zhang Z, Li D, Su Z (2022) Joint optimization of computing offloading and service caching in edge computing-based smart grid. IEEE Trans Cloud Comput 11(2):1122–1132
13. Xu Z, Zhou L, Chau SC-K, Liang W, Dai H, Chen L, Xu W, Xia Q, Zhou P (2022) Near-optimal and collaborative service caching in mobile edge clouds. IEEE Trans Mobile Comput 22(7):4070–4085
14. Hao Y, Chen M, Gharavi H, Zhang Y, Hwang K (2020) Deep reinforcement learning for edge service placement in softwarized industrial cyber-physical system. IEEE Trans Indust Inf 17(8):5552–5561

15. Sami H, Mourad A, Otrok H, Bentahar J (2021) Demand-driven deep reinforcement learning for scalable fog and service placement. IEEE Trans Serv Comput 15(5):2671–2684
16. Talpur A, Gurusamy M (2021) Drld-sp: A deep-reinforcement-learning-based dynamic service placement in edge-enabled internet of vehicles. IEEE Internet of Things J 9(8):6239–6251
17. Chen Y, Sun Y, Yang B, Taleb T (2022) Joint caching and computing service placement for edge-enabled iot based on deep reinforcement learning. IEEE Internet of Things J 9(19):19501–19514
18. Xue Z, Liu C, Liao C, Han G, Sheng Z (2023) Joint service caching and computation offloading scheme based on deep reinforcement learning in vehicular edge computing systems. IEEE Trans Veh Technol 72(5):6709–6722
19. Zhou H, Zhang Z, Wu Y, Dong M, Leung VC (2022) Energy efficient joint computation offloading and service caching for mobile edge computing: A deep reinforcement learning approach. IEEE Trans Green Commun Netw 7(2):950–961
20. Tang Q, Xie R, Fang Z, Huang T, Chen T, Zhang R, Yu FR (2024) Joint service deployment and task scheduling for satellite edge computing: A two-timescale hierarchical approach. IEEE J Select Areas Commun
21. Wei X, Cai L, Wei N, Zou P, Zhang J, Subramaniam S (2023) Joint uav trajectory planning, dag task scheduling, and service function deployment based on drl in uav-empowered edge computing. IEEE Internet of Things J 10(14):12826–12838
22. Wei Z, Li B, Zhang R, Cheng X, Yang L (2023) Many-to-many task offloading in vehicular fog computing: A multi-agent deep reinforcement learning approach. IEEE Trans Mobile Comput
23. Zhang D, Wang W, Zhang J, Zhang T, Du J, Yang C (2023) Novel edge caching approach based on multi-agent deep reinforcement learning for internet of vehicles. IEEE Trans Intell Transport Syst
24. Naderializadeh N, Sydir JJ, Simsek M, Nikopour H (2021) Resource management in wireless networks via multi-agent deep reinforcement learning. IEEE Trans Wireless Commun 20(6):3507–3523
25. Gronauer S, Diepold K (2022) Multi-agent deep reinforcement learning: a survey. Art Intell Rev 55(2):895–943
26. Zhong C, Gursoy MC, Velipasalar S (2019) Deep multi-agent reinforcement learning based cooperative edge caching in wireless networks. In: ICC 2019-2019 IEEE International Conference on Communications (ICC), pp 1–6. IEEE
27. Zhou H, Jiang K, He S, Min G, Wu J (2023) Distributed deep multi-agent reinforcement learning for cooperative edge caching in internet-of-vehicles. IEEE Trans Wireless Commun 22(12):9595–9609
28. Yao Z, Xia S, Li Y, Wu G (2023) Cooperative task offloading and service caching for digital twin edge networks: A graph attention multi-agent reinforcement learning approach. IEEE J Select Areas in Commun
29. Huang B, Liu X, Xiang Y, Yu D, Deng S, Wang S (2022) Reinforcement learning for cost-effective iot service caching at the edge. J Para Distrib Comput 168:120–136
30. Hu Z, Fang C, Wang Z, Tseng S-M, Dong M (2023) Many-objective optimization based-content popularity prediction for cache-assisted cloud-edge-end collaborative iot networks. IEEE Internet of Things J
31. Wu C, Xu Z, He X, Lou Q, Xia Y, Huang S (2024) Proactive caching with distributed deep reinforcement learning in 6 g cloud-edge collaboration computing. IEEE Trans Para Distrib Syst
32. Lin P, Ning Z, Zhang Z, Liu Y, Yu FR, Leung VC (2023) Joint optimization of preference-aware caching and content migration in cost-efficient mobile edge networks. IEEE Trans Wireless Commun
33. Castenow J, Feldkord B, Knollmann T, Malatyali M, Heide F (2020) The online multi-commodity facility location problem. In: Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures, pp 129–139
34. Yang Y, Lou K, Wang E, Liu W, Shang J, Song X, Li D, Wu J (2023) Multi-agent reinforcement learning based file caching strategy in mobile edge computing. IEEE/ACM Trans Netw
35. Hassan SS, Park YM, Tun YK, Saad W, Han Z, Hong CS (2023) Satellite-based its data offloading & computation in 6g networks: A cooperative multi-agent proximal policy optimization drl with attention approach. IEEE Trans Mobile Comput
36. Van Hasselt H, Guez A, Silver D (2016) Deep reinforcement learning with double q-learning. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol 30