Computer Modeling in Engineering & Sciences

Tech Science Press

DOI: 10.32604/cmes.2024.048759

ARTICLE

A Fault-Tolerant Mobility-Aware Caching Method in Edge Computing

Yong Ma[1], Han Zhao[2], Kunyin Guo[3,*], Yunni Xia[3,*], Xu Wang[4], Xianhua Niu[5], Dongge Zhu[6] and Yumin Dong[7]

[1]School of Computer Information Engineering, Jiangxi Normal University, Nanchang, 330000, China

[2]School of Digital Industry, Jiangxi Normal University, Shangrao, 334000, China

[3]The College of Computer Science, Chongqing University, Chongqing, 400044, China

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

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

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

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

*Corresponding Authors: Kunyin Guo. Email: css@cqu.edu.cn; Yunni Xia. Email: xiayunni@hotmail.com

Received: 17 December 2023 Accepted: 06 February 2024 Published: 16 April 2024

ABSTRACT

Mobile Edge Computing (MEC) is a technology designed for the on-demand provisioning of computing and storage services, strategically positioned close to users. In the MEC environment, frequently accessed content can be deployed and cached on edge servers to optimize the efficiency of content delivery, ultimately enhancing the quality of the user experience. However, due to the typical placement of edge devices and nodes at the network's periphery, these components may face various potential fault tolerance challenges, including network instability, device failures, and resource constraints. Considering the dynamic nature of MEC, making high-quality content caching decisions for real-time mobile applications, especially those sensitive to latency, by effectively utilizing mobility information, continues to be a significant challenge. In response to this challenge, this paper introduces FT-MAACC, a mobility-aware caching solution grounded in multi-agent deep reinforcement learning and equipped with fault tolerance mechanisms. This approach comprehensively integrates content adaptivity algorithms to evaluate the priority of highly user-adaptive cached content. Furthermore, it relies on collaborative caching strategies based on multi-agent deep reinforcement learning models and establishes a fault-tolerance model to ensure the system's reliability, availability, and persistence. Empirical results unequivocally demonstrate that FT-MAACC outperforms its peer methods in cache hit rates and transmission latency.

KEYWORDS

Mobile edge networks; mobility; fault tolerance; cooperative caching; multi-agent deep reinforcement learning; content prediction

1 Introduction

With the exponential growth of data traffic generated by Internet of Things (IoT) devices, existing networks and infrastructure are facing significant challenges [1]. The traditional cloud-based model

is increasingly proving to be inadequate, with its lack of agility and efficiency becoming particularly evident in the face of growing demands and data loads. In conventional cloud computing systems, data must traverse the Internet to reach cloud servers for processing, leading to significant latency issues, which are intolerable for real-time applications [2]. To overcome these limitations, Mobile Edge Computing (MEC) technology has emerged [3], enabling the caching of resources at the network edge and providing computing and storage services close to the requests. This enhances the responsiveness of latency-sensitive applications and ensures users experience rapid system responses. However, due to its reliance on unreliable wireless communication and distributed resource infrastructure, applications based on MEC are more susceptible to various types of system failures or faults, including MEC overload and software or hardware failures, all of which further contribute to a poor Quality of Experience (QoE) perceived by users [4].

Despite the numerous benefits of edge computing, its highly distributed and dynamic nature, coupled with the diversity of user preferences, service requests, and mobility patterns, make ensuring high cache utilization and user satisfaction through effective fault tolerance mechanisms a complex and challenging task. To address this challenge and ensure reliable and seamless content delivery to users, several key issues need to be addressed:

(i) Edge servers have limited capacity and cannot cache all the content that users might request.

(ii) Edge servers must provide collaborative services with both caching and communication functions, rather than building isolated services for individual users.

(iii) Accurately predicting user preferences, considering the differences in content preferences among individuals, is essential.

(iv) The caching mechanism should align with user mobility, ensuring that the requests of mobile users are met with guaranteed service efficiency.

(v) Edge nodes are susceptible to malicious activities, physical damage in harsh environments, or temporary disruptions in network connectivity when frequently exchanging tasks, which can impact real-time data communication.

As a response, we introduce a mobile-aware caching approach that integrates fault tolerance mechanisms. This paper primarily focuses on caching strategies aimed at reducing latency and enhancing cache hit rates and proposes a fault tolerance mechanism to optimize the performance of MEC systems, ultimately improving user experience and system efficiency. Through extensive large-scale simulation experiments, we have validated the superior performance and effectiveness of the FT-MAACC method compared to conventional approaches.

The remaining part of this paper is organized as follows: a summary of related work is provided in Section 2. The system model and problem formulation are presented in Section 3. Section 4 discusses the proposed primary methods. Simulation results will be discussed in Section 5. Finally, closing remarks and prospects are provided in Section 6.

2 Related Work

In recent years, with the advancement of MEC, content caching technology has garnered significant international research interest as an effective solution for reducing data traffic. This technology reduces latency and network congestion by enabling servers to quickly fulfill user requests through the local storage of popular content. Consequently, the research community has developed numerous MEC caching strategies. For example, Reiss-Mirzaei et al. [5] considered the social and behavioral

characteristics of users that affect caching strategies and proposed an edge caching mechanism based on social relationships and user behavior characteristics, aimed at reducing network traffic and latency in accessing popular content. Zhang et al. [6] developed a collaborative edge caching framework that integrates intelligent vehicles for multimedia services in modern wireless networks, enhancing mobile edge computing resource utilization. Ndikumana et al. [7] addressed mobile network latency issues and proposed collaborative cache allocation and computation offloading strategies to maximize resource utilization.

In the context of enhancing network performance and user satisfaction, personalized caching and mobility-aware MEC caching have garnered significant attention. These approaches are crucial in meeting user requirements, optimizing network efficiency, and providing personalized services and content recommendations. Tang et al. [8] addressed the efficient use of edge computing and IoT devices for caching multimedia content in Information-Centric Networks, proposing a solution integrating machine learning-based location prediction, intelligent caching, and optimized caching replacement algorithms. Wu et al. [9] explored collaborative caching in Vehicle Edge Computing (VEC), using asynchronous joint learning and deep reinforcement learning to optimize caching locations and improve the global model's accuracy. Wei et al. [10] introduced user destination prediction with trajectory information, integrated with a caching decision algorithm for creating efficient cache deployment plans. Given the scale and complexity of MEC caching, deep reinforcement learning algorithms, like Q-learning [11], have shown significant potential in optimizing these problems. Jiang et al. [12] delved into collaborative content caching in MEC, proposing a reinforcement learning-based architecture and a Multi-Agent Reinforcement Learning (MARL) algorithm to manage content caching, particularly for unknown user preferences inferred from historical demand patterns.

Although multi-agent deep reinforcement learning has its advantages, utilizing the shared experiences among agents to improve learning efficiency, it also has limitations, mainly due to insufficient consideration of user mobility characteristics. As Ostrowski et al. [13] proposed, in dynamic network environments, mobile fog nodes and end users exhibit time-varying characteristics, including dynamic network topology changes, which pose additional challenges to the effectiveness and applicability of deep reinforcement learning algorithms. In contrast, Zhong et al. [14] proposed a system model encompassing both centralized and decentralized caching systems, introducing a deep learning-based Actor-Critic framework to optimize content delivery latency. Their model, however, is static, differing from the dynamic approach in this paper. Additionally, Song et al. [15] focused on static users within MEC environments, using a single-agent learning mechanism to optimize collaborative caching models. Single-agent deep reinforcement learning focuses on individual learning in isolated environments, not fully exploiting the collaborative potential among multiple agents, which is a key focus of multi-agent deep reinforcement learning in this study.

In the distributed and heterogeneous environment of edge computing, where numerous edge nodes and devices are present, risks such as hardware malfunctions, network disruptions, or power outages are common. Given the inevitability of failures in this environment, fault tolerance becomes a crucial aspect. Consequently, researchers have developed various strategies to address this challenge. For instance, Zhang et al. [16] introduced an online offloading framework utilizing multi-step reinforcement learning, featuring fault detection, recovery strategies, backup-based fault recovery, and checkpoint techniques. Sun et al. [17] proposed a fault-tolerant Quality of Service (QoS)-aware scheduling model for mixed-edge-cloud environments, evolving the traditional Primary-Backup (PB) model into a reliability-oriented time-constrained approach. Concurrently, Mitsis et al. [18] presented a multi-user-multi-server-multi-access edge computing operational framework based on prospect theory, game theory, and reinforcement learning principles. Their framework focuses on

enabling MEC servers to determine their optimal published prices in a semi-autonomous and fully autonomous manner, marking a significant advancement in our research field. However, compared to our proposed FT-MAACC, Mitsis et al.'s study primarily concentrates on pricing strategies and behavioral awareness, while our work is more focused on integrating content adaptability algorithms, collaborative caching strategies, and fault tolerance models to address fault tolerance challenges in the MEC environment.

Our approach combines a multi-agent deep reinforcement learning (DRL) model with collaborative caching strategies, integrating fault tolerance mechanisms to address base station failures. In the dynamic landscape of distributed edge computing, where user and base station mobility frequently alter network conditions [19], a single-agent learning approach might struggle to keep pace with these changes. This is where the advantage of multiple DRL agents becomes evident. They independently explore and learn in diverse environments, effectively adapting to these shifts, meeting user demands more efficiently and enhancing overall network performance. Unlike single-agent methods, multi-agent DRL not only offers greater potential in adapting to such dynamic environments but also brings forth new challenges in cooperation and communication, which are vital for optimizing content caching and delivery processes. The integration of fault tolerance mechanisms concurrently plays a crucial role, in mitigating the effects of base station failures and ensuring the stability and reliability of the system.

3 System Model and Problem Formulation

In this section, we provide a detailed exposition of the system model and the formulation of the problem.

3.1 System Model

In this paper, we deploy an MEC environment where each base station is equipped with an edge server with a storage capacity denoted as $\mathcal{T}$. Each mobile user establishes a connection with a specific MEC server, and within any given time slot, a user can only communicate with one base station. MEC servers are interconnected with each other through X2 or Xn links and connected to the cloud via backhaul links [20]. The set of base stations is denoted as $BS = \{bs_1, bs_2, bs_3, \ldots, bs_i\}$, the set of users is denoted as $U = \{u_1, u_2, u_3, \ldots, u_j\}$, and the cloud server serves as the source server for content. It is assumed that content requests are mutually independent, and each user can request only one content item within a specific time slot, with user locations remaining fixed during the given time slot. The system model comprises four key components: the cache model, transmission delay model, mobility model and fault model. This comprehensive framework is designed to address user demands in high-mobility environments, with the goals of reducing user request latency, improving cache hit rates, and optimizing MEC system performance through fault tolerance mechanisms. Reference to Table 1 is provided for annotations in this paper.

Table 1: Summary of the key notations

Notation	Definition
$\mathcal{T}$	The capacity of each edge server
$t \in T$	Time slot
$BS = \{bs_1, bs_2, bs_3, \ldots, bs_i\}$	Set of base stations

(Continued)

Table 1 (continued)

Notation	Definition
bs_k	The k-th base station, $k \in [1, i]$
$U = \{u_1, u_2, u_3, \ldots, u_j\}$	Set of users
u_b	The b-th user, $b \in [1, j]$
$C = \{c_1, c_2, c_3, \ldots, c_n\}$	The set of content
c_p	The c-th content, $p \in [1, n]$
$w^t_{bs_k, u_b, c_p}$	The content c_p requested by user u_b from base station bs_k at t
$\Pi^t_{k,p}$	The caching status of c_p at bs_k
ξ_p	The size of p_{th} content
$D^t_{k,local}$, $D^t_{k,neighbor}$, $D^t_{k,cloud}$	Delay from local, neighbour and cloud server
D^t	Total transmission delay
$L^t_{u_b}$	The moving trajectory of the b-th user
PF_k	Failure probability of the k-th base station
TF_k	The probability of the k-th base station failing to process tasks
FT_K	Delay caused by failure of the k-th base station
S^t, Ag^t, A^t	The set of state, agent, action
$R(S^t, A^t)$	Reward function
κ	Content type
$f_\kappa(t_m)$	Access content
$R_{f_\kappa}(t_m)$	Corresponding content's request count
$P^{tm}_{f_\kappa}$	Content fitness
$Cha_\kappa(t_m)$	Retrieve the relevant attributes of the content
S_κ	Content feature adaptation degree

As illustrated in Fig. 1, the system architecture can be divided into three fundamental layers [21]. At the topmost tier is the cloud server layer, consisting of high-performance servers with ample storage capacity. The intermediate layer is the edge server layer, comprising various base stations, each equipped with data storage, caching capabilities, and computational resources. The bottom layer is composed of mobile vehicle users. These base stations are distributed randomly within the simulated area, interconnected through wireless networks, and linked to the cloud servers via backhaul connections. When a user initiates a content request, the request is initially transmitted to the nearby base station. The base station then searches its local cache for the requested content. If the content is available in the local cache, the base station promptly delivers it to the mobile user. In cases where the desired content is not found in the local cache, the base station attempts to request the content from neighboring base stations. Only if adjacent base stations cannot provide the requested content does the base station forward the request to the cloud center [22].

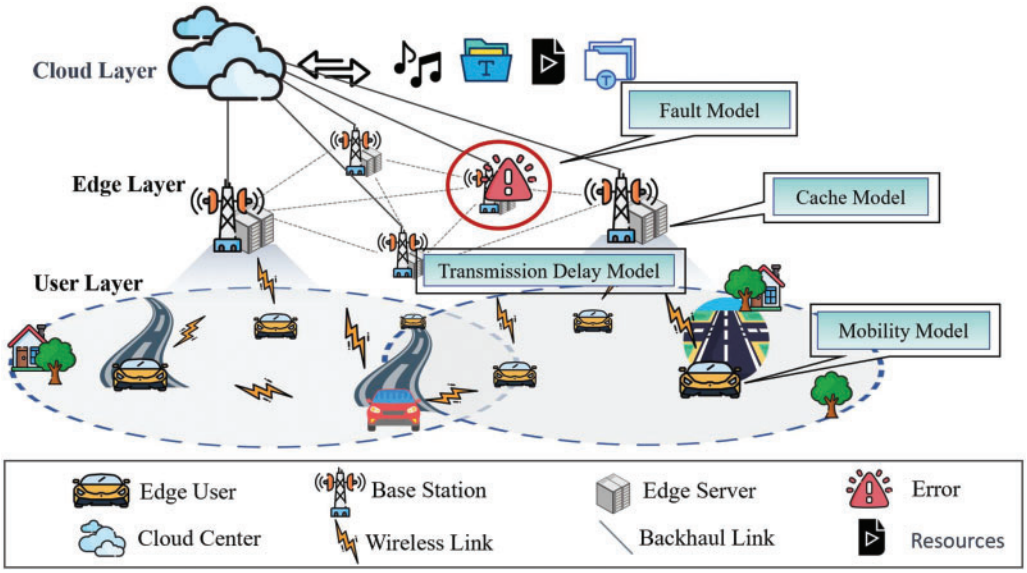

Figure 1: Edge computing system model

3.2 Cache Model

Edge servers face limitations related to storage capacity and computational resources, which necessitate the selective caching of content to ensure quality of service for users [23]. At t, the cache status of edge server bs_k can be represented as:

$$C_k(t) = \left\{\Pi_{k,1}^t,\ \Pi_{k,2}^t, \ldots,\ \Pi_{k,m}^t\right\} \tag{1}$$

where, the binary variable $\Pi_{k,j}$ is used to denote whether content c_p is cached in bs_k, with $\Pi_{k,p}$ defined as:

$$\Pi_{k,p}^t = \begin{cases} 0, & \text{if } c_p \text{ is cached in } bs_k \\ 1, & \text{otherwise} \end{cases} \tag{2}$$

Considering the constraints imposed by hardware resources and storage capacity, the cache capacity of edge servers should not exceed their total available space. Hence, edge servers must adhere to Eq. (3). This implies that, at t, if content c_w is to be cached in bs_k, it is necessary for the current server's capacity to be sufficiently available, as defined by the equation:

$$\sum_{p=1,p\neq w}^{n} \Pi_{k,p}^t \cdot \xi_p + \xi_w \leq \mathcal{T} \tag{3}$$

where, ξ_p represents the size of content c_p and T_k represents the cache capacity of bs_k.

3.3 Transmission Delay Model

Transmission delay refers to the time delay experienced by data during its transfer across a network. This delay can be categorized into three primary components: firstly, the delay associated with data transmission within local caches; secondly, the delay arising from collaborative transfers between various edge servers; and lastly, the delay incurred when data is transmitted from the cloud to end users. These three components collaboratively contribute to the reduction of the overall

transmission delay [24]. In t, the three components of transmission delay for bs_k are defined as $D^t_{k,local}$, $D^t_{k,nighbor}$, and $D^t_{k,cloud}$, and their specific mathematical expressions are as follows:

$$D^t_{k,local} = \sum_{p=1}^{n} \sum_{b=1}^{j} w^t_{bs_k,u_b,c_p} \cdot \xi_p \cdot \Pi^t_{k,p} / r_{local} \tag{4}$$

$$D^t_{k,neighbor} = \sum_{p=1}^{n} \sum_{b=1}^{j} w^t_{bs_k,u_b,c_p} \cdot \xi_p \cdot \left(1 - \Pi^t_{k,p}\right) \cdot \Pi^t_{N_k,p} / r_{neighbor} \tag{5}$$

$$D^t_{k,cloud} = \sum_{p=1}^{n} \sum_{b=1}^{j} w^t_{bs_k,u_b,c_p} \cdot \xi_p \cdot \left(1 - \Pi^t_{k,p}\right) \cdot \left(1 - \Pi^t_{N_k,p}\right) / r_{cloud} \tag{6}$$

where, ξ_p represents the size of content c_p, $w^t_{bs_k,u_b,c_p}$ represents the content c_p requested by user u_j from base station bs_k at time t, r_{local}, $r_{neighbor}$, r_{cloud} represent the transmission rates from the local, neighbor, and central servers, respectively. Π_p indicates whether content c_p is cached at the current base station, and $\Pi^t_{N_k,p}$ indicates whether content c_p is cached at neighboring nodes.

The total transmission delay is defined as:

$$D^t = \sum_{k=1}^{i} D^t_{k,local} + D^t_{k,neighbor} + D^t_{k,cloud} \tag{7}$$

3.4 Mobility Model

In this assumption, we consider that the mobility of a mobile user exhibits an arbitrary pattern, with the direction and angle of movement varying over time. This implies that the user's movement trajectory is not constrained to a specific pattern or path and can change dynamically based on various factors or influences. Thus the user's path is expressed in terms of latitude and longitude as follows:

$$L^t_{u_b} = \{lon_{u_b}(t), lat_{u_b}(t)\} \tag{8}$$

$L^t_{u_b}$ denotes the moving trajectory of the b_{th} user in t. Where $lat_{u_b}(t)$ denotes the longitude point of the mobile trajectory of the b_{th} user in t, and $lon_{u_b}(t)$ denotes the latitude point of the mobile trajectory of the b_{th} user in t.

3.5 Fault Model

The failure probability of edge servers is a multifaceted issue that is influenced by various factors. The failure probability is affected by multiple factors, such as the quality of hardware equipment, environmental conditions, operating time, and workload status [25]. This paper assumes that the failure events of edge servers are independent and follow a Poisson process. Consequently, the failure probability of an edge server equipped with base station bs_k in a single time slot, denoted as $PF_k(X)$, can be further elaborated as:

$$\begin{aligned} PF_k(X) &= \frac{(e^{\lambda t} - 1)\, \mathrm{f_k}^x}{x!\, e^{\lambda t} \mathrm{T}_k} \\ 0 &\le PF_k(X) \le 1 \\ x &\in \{0, 1, 2, \ldots, n\} \end{aligned} \tag{9}$$

where X represents the number of failure occurrences, e is the base of the natural logarithm, λ is the failure rate (the average rate of failure events), f_k represents the number of failures in base station bs_k, and T_k is the time during which bs_k failures occur.

The probability of task processing failure $TF_k(X)$ on the edge server bs_k, when a user requests resources, can be estimated using queuing theory:

$$TF_k(X) = \frac{\rho^{CN(1-\rho)}}{e^{\rho \mathrm{t}} \mathrm{T}_k} \tag{10}$$

where ρ represents the system's utilization rate (the ratio of task arrival rate to task processing rate), and CN denotes the capacity of the server.

When a base station fails, it ceases to accept user request tasks. However, if a failure occurs after receiving a user request, resulting in the inability to process the received task, it leads to a delay in task execution:

$$FT_k^t = \sum_{k=1}^{i} \frac{w^t_{bs_k,u_b,c_p} \cdot \xi_{\mathrm{p}} \cdot d_{bs_k,u_b}}{Cap_k} (1 + TF_k(X)) \tag{11}$$

where Cap_k the processing capacity of the k-th base station and d_{bs_k,u_b} denotes the distance of communication between the base station and the b-th user.

3.6 Problem Formulation

The content caching problem involves minimizing content delivery latency in scenarios that consider additional requirements or constraints. Our goal is to minimize latency to the greatest extent possible while meeting the time constraints and capacity limits of each MEC, all the while ensuring the effective delivery of the required content. Consequently, this problem is formally formulated as:

$$\begin{aligned} &\mathrm{Min}\ \frac{1}{T} \sum_{t=1}^{T} (D^t + FT_k) \\ &s.t. \quad \mathbf{C1.}\ \ \textstyle\sum_{p=1}^{n} \Pi_{k,p} \cdot\ \xi_p \le\ \mathcal{T} \\ &\qquad \mathbf{C2.}\ \ \Pi_{k,p} \in \{0, 1\} \\ &\qquad \mathbf{C3.}\ \ 0 < d_{u_b} \le\ d_{max} \\ &\qquad \mathbf{C4.}\ \ FT_k < T_{\text{neighbor}} \end{aligned} \tag{12}$$

Constraint C1 indicates the size of the content requested by the user cannot exceed the capacity size of the edge server. Constraint C2 is to constrain the non-negativity and integrity of the variable. Constraint C3 indicates the maximum distance of the user request content. Constraint C4 states that the delay caused by a base station failure, leading to the inability to process an individual user request task, must be shorter than the time it takes for a user to successfully send a request to the base station.

4 Primary Methods

4.1 Content Prediction

To fulfill user requests and minimize content transmission latency, edge servers can proactively predict the content that users are likely to request and cache it locally. While highly popular content is more likely to be requested, it fails to account for users' personal preferences [26]. Therefore, simply predicting based on popularity is not always realistic. In the context of users' mobility, accurately predicting content that aligns with their preferences and caching the target content accordingly is a

challenging problem that aims to further enhance the QoS for users. Based on this, we propose an adaptive content prediction algorithm (ACP) that takes into account user preferences. The specific algorithm is depicted in Algorithm 1.

Algorithm 1: ACP algorithm

1: κ: Content type
2: $f_\kappa\ (t_m)$: Access content
3: $R_{f_\kappa}(t_m)$: Number of requests for the corresponding content
4: $P_{f_\kappa}^{t_m}$: Content fitness
5: $Cha_\kappa(t_m)$: Retrieve the relevant attributes of the content
6: S_κ: Content feature adaptation degree
7: **for** each $u_i \in$ Users **do**
8: $P_{f_\kappa}^{t_\mathrm{m}} = \dfrac{R_{f_\kappa}(t_\mathrm{m})}{\sum_\kappa R_{f_\kappa}(t_\mathrm{m})}$
9: Predict $\hat{P}_{f_\kappa}^{t_m+1}$ from $\left\{P_{f_\kappa}^{t_0}, \quad P_{f_\kappa}^{t_1}, \ldots, P_{f_\kappa}^{t_\mathrm{m}}\right\}$
10: $\hat{P}_{f_\kappa}^{t_m+1}$ for $f_\kappa\ (t_m+1)$
11: Predict $\widehat{f_\kappa}(t_\mathrm{m}+1)$
12: Predict $\widehat{\mathrm{Cha}_\kappa}(t+1)$ from $\widehat{f_\kappa}(t_\mathrm{m}+1)$
13: Calculate $S_\kappa = \sqrt{\sum_\kappa \theta_\kappa(\widehat{Cha_\kappa}(t_m+1) - Cha_k(t_m))^2}$
14: Calculate $P_{f_\kappa}(t_m+1) = \dfrac{\sum_{z=0}^{N} S_{\kappa,z}, P_{f_\kappa,z}^{t_m}}{N}$
15: Uploads $P_{f_\kappa}^{t_m}$ to the local BS
16: **end for**
17: Content suitability assessment and selection for inclusion in CN
18: **return** CN

Initially, collect the historical records of user content access within time period T. The cumulative number of content accessed by users is denoted as W. Time period T is then divided into o time slots, denoted as $\{t_0, t_1, \ldots, t_\mathrm{o}\}$. For each time t_m $(0 < \mathrm{m} < \mathrm{o})$, calculate the content fitness $P_{f_\kappa}^{t_\mathrm{m}}$ of user content access. The calculation formula is as follows:

$$P_{f_\kappa}^{t_\mathrm{m}} = \frac{R_{f_\kappa}(t_\mathrm{m})}{\sum_\kappa R_{f_\kappa}(t_\mathrm{m})} \tag{13}$$

where, κ represents the category of content accessed by users, f_κ represents the content accessed by users during time t_m, and $R_{f_\kappa}(t_\mathrm{m})$ indicates the number of requests made by users for content $f_\kappa\ (t_\mathrm{m})$ during time t_m. By inputting the collection of content matching degrees $\left\{P_{f_\kappa}^{t_0}, P_{f_\kappa}^{t_1}, \ldots, P_{f_\kappa}^{t_\mathrm{m}}\right\}$ into an exponential smoothing prediction model, we can obtain the predicted content matching degree $\hat{P}_{f_\kappa}^{t_\mathrm{m}+1}$ for users in the next time slot. Consequently, the predicted content $\widehat{f_\kappa}\ (t_\mathrm{m}+1)$ for user clusters in that time slot can be determined.

The content feature adaptation degree S_κ represents the degree of match between the predicted content $\widehat{f_\kappa}\ (t_\mathrm{m}+1)$ for users in time $t_\mathrm{m}+1$ and their accessed content $f_\kappa\ (t_\mathrm{m})$ in time t_m. The calculation

formula is as follows:

$$S_\kappa = \sqrt{\sum_\kappa \theta_\kappa \left(\widehat{\mathrm{Cha}_\kappa}\left(t_\mathrm{m}+1\right) - \mathrm{Cha}_\kappa\left(t_\mathrm{m}\right)\right)^2} \tag{14}$$

where θ_κ represents the weights of different features. $\mathrm{Cha}_\kappa\left(t_m\right)$ denotes the corresponding feature of the accessed content $f_\kappa\left(t_\mathrm{m}\right)$ by users in time t_m. $\widehat{\mathrm{Cha}_\kappa}\left(t_\mathrm{m}+1\right)$ represents the corresponding feature of the predicted content $\widehat{f_\kappa}\left(t_\mathrm{m}+1\right)$ based on the content adaptation degree prediction value $\hat{P}_{f_\kappa}^{t_\mathrm{m}+1}$ for users in $t_\mathrm{m}+1$.

Based on the content feature matching, the content adaptation degree $P_{f_\kappa}^{t_\mathrm{m}+1}$ for predicting the content of the user cluster at $t_\mathrm{m}+1$ is calculated using the following formula:

$$P_{f_\kappa}^{t_\mathrm{m}+1} = \frac{\sum_{z=0}^{Z} S_{\kappa,z} \cdot P_{f_{\kappa,z}}^{t_\mathrm{m}}}{N} \tag{15}$$

where, $S_{\kappa,z}$ represents the content feature matching degree of the κ_{th} content category for the z_{th} user cluster; $P_{f_{\kappa,z}}^{t_\mathrm{m}}$ represents the content adaptiveness of the z_{th} user cluster for the κ_{th} content category in t_m.

4.2 Multi-Agent Deep Reinforcement Learning Model

DRL is a technique that merges deep learning and reinforcement learning to train intelligent agents [27]. By engaging in ongoing interactions with the environment, these agents learn optimal strategies to make decisions. Multiple DRL agents refer to the presence of multiple independent intelligent agents simultaneously in a reinforcement learning environment [28]. In traditional reinforcement learning, typically only one agent interacts with the environment and learns the optimal strategy. The objective of multiple deep reinforcement learning agents is to learn and optimize their strategies through collaboration or competition, aiming to achieve global optimality or specific goals. Each agent perceives the environmental state, selects actions, and interacts with the environment, continuously improving its strategy through trial and error and learning [29].

In this paper, we use a multi-intelligent deep reinforcement learning framework (Fig. 2), and partially incomplete observable reinforcement learning problems can be modeled as Partially Observable Markov Decision Processes (POMDP) [30] 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.

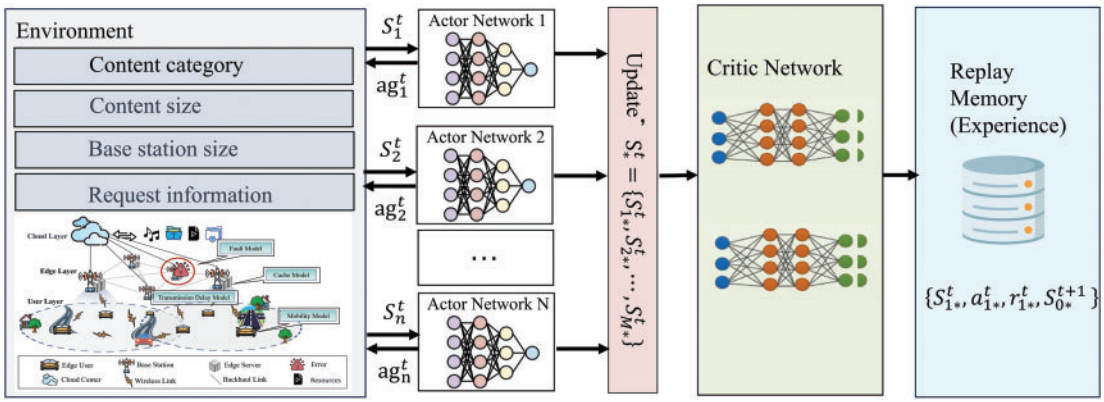

Figure 2: Schematic illustration of the Multi-Agent Actor-Critic framework

$$P = \begin{bmatrix} P_{11} & \cdots & P_{1h} \\ \vdots & \ddots & \vdots \\ P_{h1} & \cdots & P_{hh} \end{bmatrix}$$

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: at time t, the specific state of the edge cache nodes is denoted as $S^t = \left\{S_1^t, S_2^t, \cdots, S_M^t\right\}$. The set of agents is defined as $Ag^t = \left\{ag_1^t, ag_2^t, \ldots, ag_N^t\right\}$, where N represents the number of agents. Each agent is placed on a specific edge server device and engages in interactions and competitions with other agents to enhance its strategy. Through continuous learning, the agents strive to improve their performance over time.

2) Action: The action set $A^t = \left\{a_s^t, a_f^t\right\}$ represents the decisions made by the agent based on its perception of the environmental state regarding whether to cache content. In this context, a_s^t denotes the action of caching the content, while a_f^t represents the action of discarding the content.

3) Reward: The reward function $R(S^t, A^t)$ represents the reward obtained by the agent for selecting a particular action in the current state. The agent utilizes this reward to adjust its exploration strategy. The general procedure involves the agent first interacting with the environment to obtain the state S^t. Based on the input state S^t, the agent selects an action A^t and receives an immediate reward. Finally, the agent accumulates the previous immediate rewards to obtain the current cumulative reward. The primary objective of the reward function is to minimize content delivery latency. When an action meets the requirements, the reward value increases, while it decreases when the action fails to meet the requirements. The reward function is directly tied to the optimization objective and can be defined as follows:

$$R\left(S^t, A^t\right) = \begin{cases} e^{-\min\left(\frac{1}{T}\sum_{t=1}^{T}(D^t + FT_k)\right)}, & \boldsymbol{a}_s^t \\ -1, & \boldsymbol{a}_f^t \end{cases} \tag{16}$$

4.3 Fault-Tolerant Multi-Agent Actor-Critic Algorithm for Content Caching

Algorithm 2, referred to as ***FT − MAACC***, innovatively integrates fault tolerance with multi-agent reinforcement learning to enhance collaboration and learning among agents within distributed environments, while simultaneously increasing the system's resilience to faults and errors. In traditional multi-agent reinforcement learning settings, failures in one or more agents can significantly impact the overall performance and stability of the system. ***FT − MAACC*** addresses this challenge by incorporating fault-tolerance mechanisms that enable the system to adapt and continue learning in the face of such failures. At the heart of this algorithm is the Actor-Critic architecture, which assigns the responsibility of decision-making based on the current state to the Actor, and the task of evaluating the outcomes of actions to the Critic. This division of responsibilities ensures that each agent in the system is equipped with its Actor and Critic, thereby addressing the critical challenge of maintaining system integrity and performance amidst potential agent failures.

Understanding the computational complexity of ***FT − MAACC*** is crucial for assessing its applicability across multi-agent systems of varying scales. The complexity is primarily influenced by several key factors, including the number of neurons in each layer of the Actor and Critic networks, the number of layers, and the dimensions of the input and output layers. For instance, the total computational complexity of the Actor-network can be expressed as $O\left(d_s \times h_{1,1} + \sum_{m=1}^{w_1-1} h_{1,m} \times h_{1,m+1} + h_{1,w_1} \times \iota\right)$, where d_s represents the size of the input layer, $h_{1,m}$ denotes the number of neurons in the m-th layer, and ι is the

dimension of the output layer. Similarly, the Critic network, which is tasked with evaluating the value of selected actions, follows a parallel complexity expression as $O\left(d_s \times h_{2,1} + \sum_{m=1}^{w_2-1} h_{2,m} \times h_{2,m+1} + h_{2,w_2}\right)$. When considering n agents within the system, the overall computational complexity becomes n times the sum of the complexities of the Actor and Critic networks for all agents.

Beyond its architectural design, ***FT − MAACC*** proactively estimates the probabilities of base station malfunctions and task processing failures to ensure fault tolerance. This assessment enhances the system's reliability and stability. By understanding the likelihood of base station outages and the risks associated with task processing errors, ***FT − MAACC*** can implement targeted strategies to mitigate potential failure scenarios. This approach contributes to reduced content delivery latencies and an improved overall user experience within edge computing environments.

Algorithm 2: FT-MAACC algorithm

Require: Number of iterative rounds T, learning rate δ, discount factor γ, actor-critic network structure.

Initialization: Initialize system parameters, hyperparameters, edge server and mobile user cache spaces, network parameters $\boldsymbol{\theta}_k, \phi_k$, target networks $\boldsymbol{\theta}'_k \leftarrow \boldsymbol{\theta}_k, \boldsymbol{\phi}'_k \leftarrow \boldsymbol{\phi}_k$, and a random process for action exploration.
if edge server bs_i is operational **then**
  Perform caching tasks.
  for each episode **do**
    for $t = 1$ to T **do**
      Randomly generate $q \in [0, 1]$.
      for each agent k in N agents **do**
        Choose actions ag_k^t based on policy or randomly for exploration.
        Gather local observations and states S_k^t.
      end for
      Interact with the environment to obtain rewards $\bar{\Delta}(t)$ and new states S_{k*}^t.
      $t = t + 1$.
    end for
    Store the events for each agent.
    Update the target network parameters as per a predefined formula (not specified here).
    Update content properties and cache states.
    Update network parameters $\boldsymbol{\theta}_k, \phi_k$.
  end for
else
    Change the state of edge server bs_i.
    Select the next base station bs_{i+1} for task processing.
    Queue tasks at bs_{i+1} for processing.
end if

Ensure: Optimal strategy Ω.

5 Performance Evaluation

5.1 Parameters Setting

To assess the FT-MAACC method and simulate content requests and user behavior in mobile environments, we integrated mobile user access event logs and mobility trajectories from the *Shanghai Telecom* dataset [31] with user content preferences from the *Movielens 1M* [32]. The Shanghai dataset

comprises over 7.2 million content access event records and their corresponding mobility trajectories from 9,481 mobile users across 3,233 edge sites over six months, reflecting users' mobility patterns and behavioral habits. By incorporating rating information from the MovieLens dataset, we attributed specific content preferences to these mobility trajectories, simulating users' interests in movies or video content at various locations and times. This approach enabled the construction of a comprehensive model that reflects both the realities of the mobile environment and user preferences. Fig. 3 shows the distribution location of edge nodes in Shanghai. Fig. 4 is an example recording the trajectory of a taxi in the city heart of Shanghai. We use Python to implement the proposed ***FT − MAACC*** method. The neural network model is built upon a network of evaluated actors and commenters, as well as a network of target actors and commenters. The target network is updated with the Adam optimizer, with learning rates of 0.001 for the critic network and 0.0005 for the actor network, and a discount factor of 0.9. The relevant parameters are shown in Table 2.

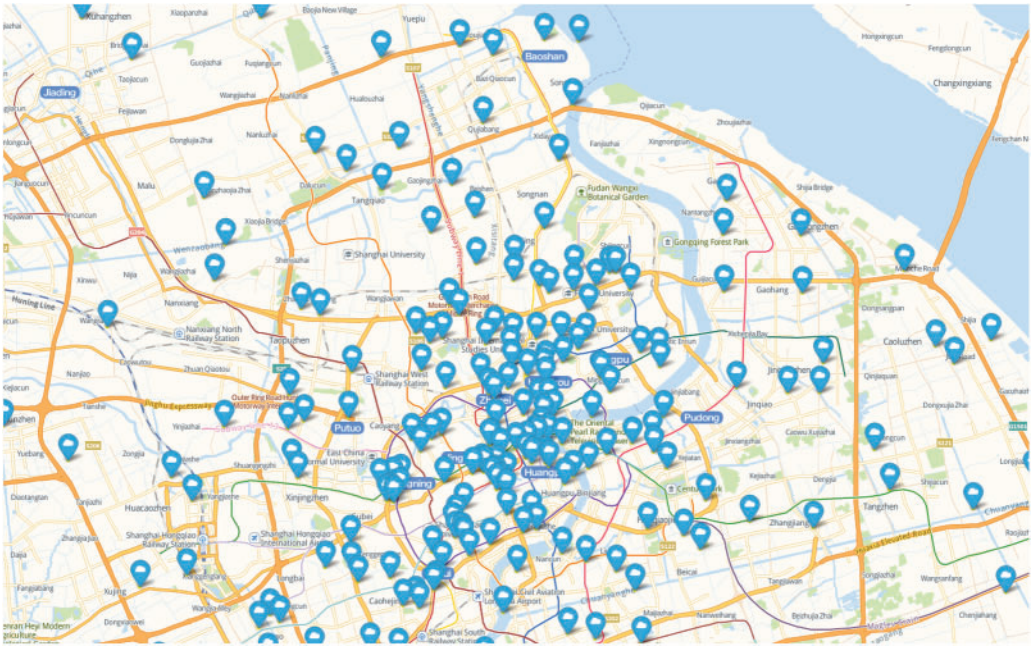

Figure 3: Blue coordinate icons represent each edge node in Shanghai's downtown area showing the general distribution of the whole network

5.2 Comparison Algorithms

FT − MAACC is compared against the following benchmark algorithms: Thompson sampling (***TS***) [33], Random Selection Algorithm (***RSA***) [34], Greedy Algorithm (***GA***) [35], ***MARL*** [12], and ***MAACC*** (The algorithm in this paper does not consider fault tolerance).

5.3 Performance Analysis

For performance analysis, we configured 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.

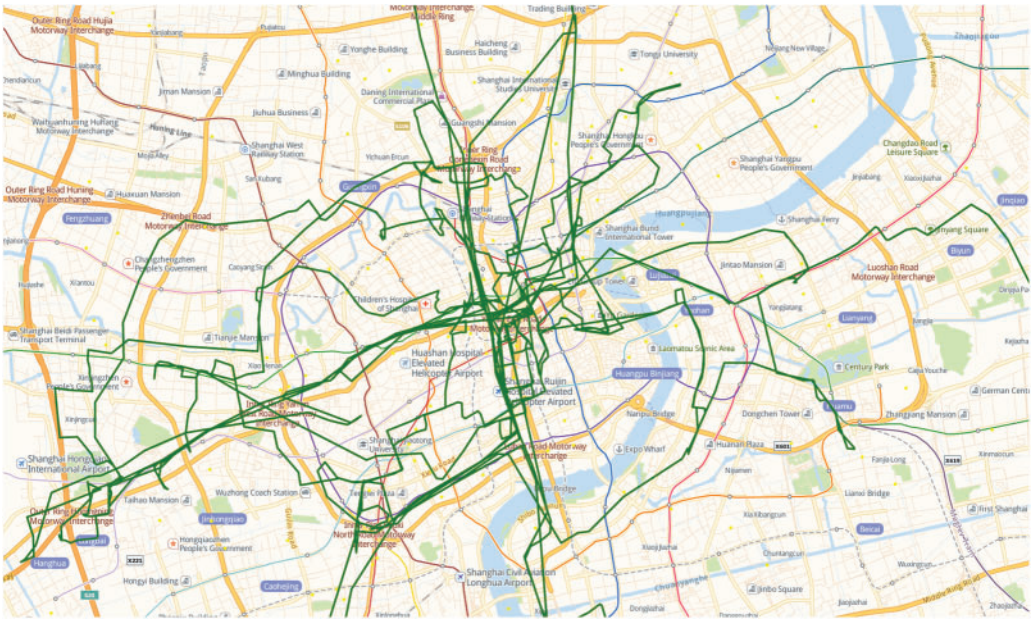

Figure 4: The green lines represent the route trajectory of a taxi in the city heart of Shanghai on a certain day

Table 2: 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. Crossed-out base stations indicate a failure in the system, while dashed lines represent the inability to acquire

resources from the faulty base stations. The ***FT − MAACC*** algorithm proactively detects base stations with a high probability of failure, selecting the nearest operational base station to provide resources. This approach aims to enhance the hit rate and reduce latency.

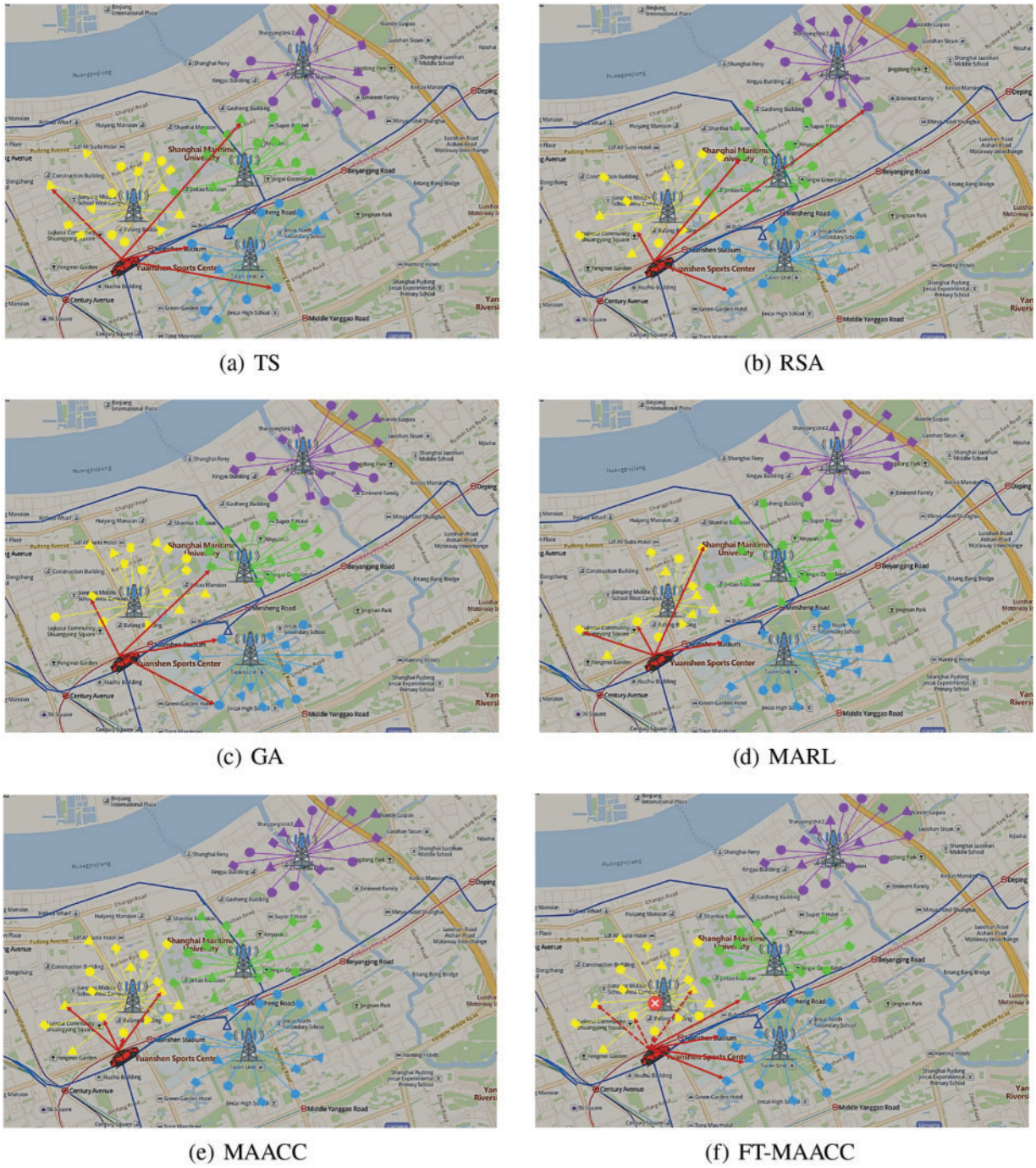

Figure 5: Cached resources using different algorithms

Fig. 6 shows the cache hit rates under various caching strategies and cache capacities in a fault-free base station scenario. As the capacity of the edge server increases, the cache hit ratio is observed to increase. By increasing the cache capacity of the edge server, the number of cached resources also increases. Consequently, mobile users are more likely to retrieve resources from local and adjacent edge servers, enhancing overall resource availability and accessibility. This results in a higher hit ratio. Since ***FT − MAACC*** solely incorporates fault tolerance mechanisms in comparison to ***MAACC***, the cache hit rates of ***FT − MAACC*** and ***MAACC*** are identical in the scenario where the base station is free from faults. ***FT − MAACC*** and ***MAACC*** outperform ***TS/RSA/GA/MARL*** by 22.2%/37.1%/15.2%/12.8%, respectively. This is because ***FT − MAACC*** and ***MAACC*** utilize multi-depth reinforcement learning agents and introduce mechanisms of cooperation and competition, allowing agents to compete, cooperate, or collaborate in their learning process. This promotes mutual learning and interaction among the agents, thereby improving overall performance.

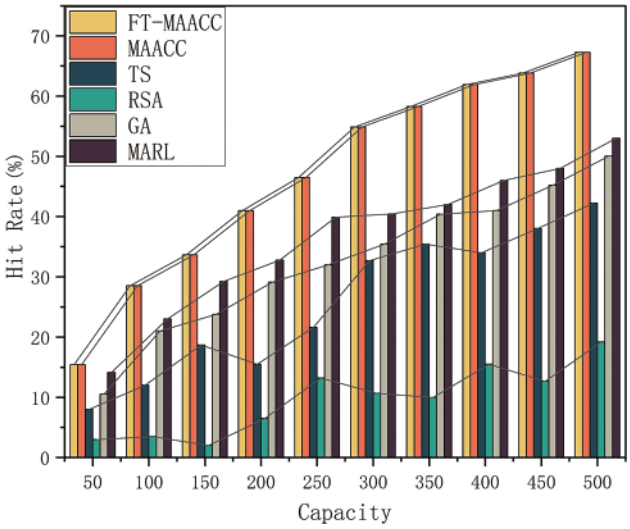

Figure 6: Cache hit rates of FT-MAACC/MAACC/TS/RSA/GA/MARL under different cache capacities (Absence of Failures)

Fig. 7 showcases the cache hit rate considering different cache capacities, various cache policies, and the occurrence of a random node failure. By incorporating fault tolerance mechanisms, ***FT − MAACC*** considers the probability and impact of node failures, giving priority to resource requests from base stations that are less susceptible to failures. This approach effectively mitigates cache misses caused by node failures, leading to an improved cache hit rate. The hit rates of other algorithms showed fluctuations primarily due to the inherent randomness of cache misses when user requests need to access cache data located on a failed node. ***FT − MAACC*** outperformed ***MAACC/TS/RSA/GA/MARL*** by 8.3%/18.4%/37.4%/17%/11.9%, respectively.

Fig. 8 reveals the request latency with different caching strategies under different cache capacities in the scenario where the base station is fault-free. An increase in the cache capacity of an edge server leads to a decrease in resource transfer latency for all cache strategies. This observation suggests a direct relationship between cache capacity and the reduction of latency in resource transfers. This is because the larger the cache capacity, the higher the cache hit ratio of the edge server, the higher the possibility of mobile users obtaining resources from the local and neighboring servers, and the lower the corresponding resource transmission delay. Based on Fig. 8, it is evident that both ***FT − MAACC***

and ***MAACC*** demonstrate comparable and lower latency compared to other methods. This can be attributed to the adoption of a multi-agent mechanism in ***FT − MAACC*** and ***MAACC***, which enhances the exploration capabilities of the training environment by incorporating diverse intelligent agents. Each agent can independently explore the environment and acquire unique strategies. This comprehensive exploration approach enables the system to discover optimal strategies and effectively reduce latency.

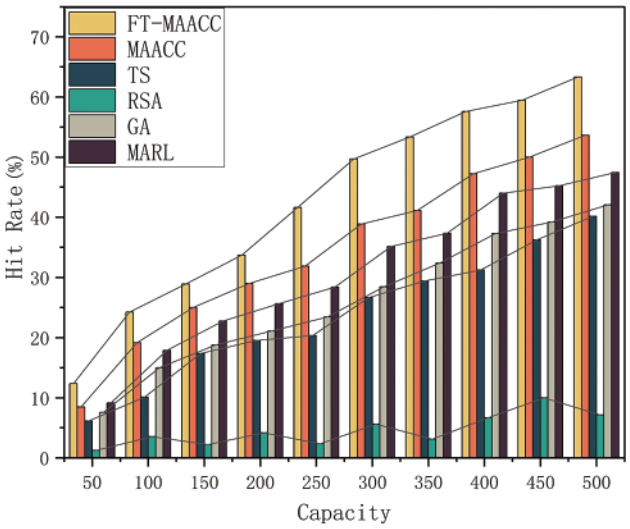

Figure 7: Cache hit rates of FT-MAACC/MAACC/TS/RSA/GA/MARL under different cache capacities (Presence of Failures)

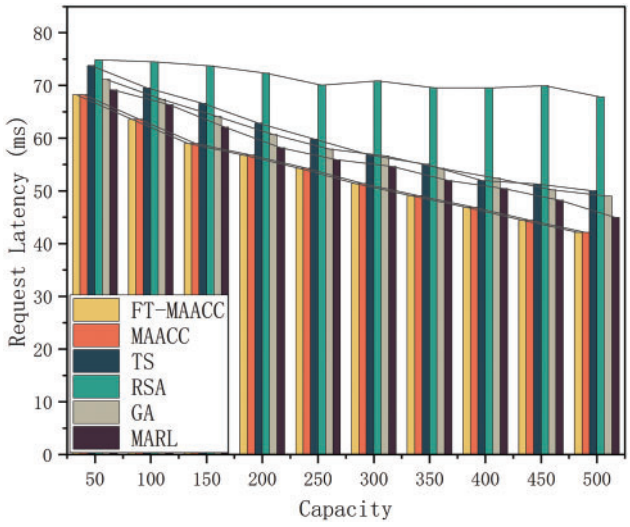

Figure 8: Request latency of FT-MAACC/MAACC/TS/RSA/GA/MARL under different cache capacities (Absence of Failures)

Fig. 9 depicts the request latency in the scenario where a random node failure occurs, considering different cache capacities and cache policies. When an edge node experiences a failure, the access

latency for users to that node increases. Among the evaluated strategies, ***FT − MAACC*** exhibits the lowest latency. However, its latency is higher compared to the scenario where no failures occur. This is primarily due to the time required for detecting and diagnosing faulty nodes. In contrast, the latency of other strategies generally increases and exhibits fluctuations. This can be attributed to the possibility of user requests being directed to the failed nodes, necessitating redirection to alternative available nodes, which introduces additional delays in network communication and forwarding. Moreover, the unavailability of data stored on the faulty nodes necessitates alternative means of data retrieval, further contributing to increased access latency. In summary, ***FT − MAACC***, with its integrated fault tolerance mechanisms, effectively mitigates latency to a certain extent, positioning it as a superior choice compared to other algorithms.

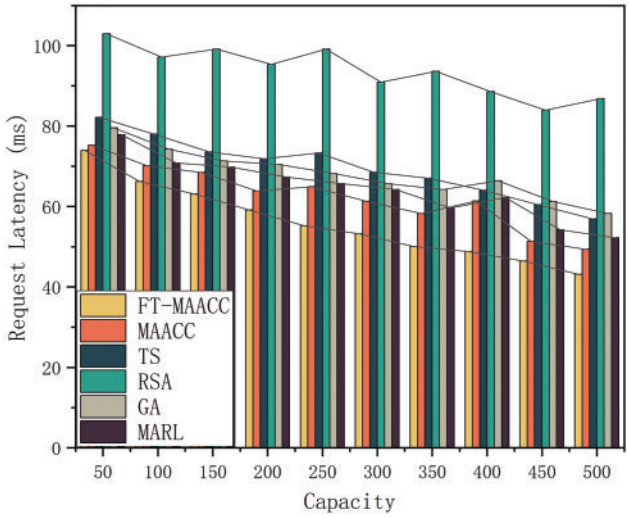

Figure 9: Request latency of FT-MAACC/MAACC/TS/RSA/GA/MARL under different cache capacities (Presence of Failures)

6 Conclusion

This article primarily addresses the challenge of high-speed mobile resource caching in the MEC environment and introduces a mobile-aware caching approach based on multi-agent deep reinforcement learning with fault tolerance (FT-MAACC). The strength of this approach lies in the agents' capacity to adapt flexibly to changing conditions and real-time feedback, autonomously updating their behaviors and strategies to suit new contexts, thus providing higher-quality decision-making and services. Experimental results illustrate the effectiveness of this approach in enhancing cache hit rates and reducing content transmission latency.

In future research and development, we plan to further fortify the reliability and fault tolerance of edge computing cache systems. By introducing more robust fault detection and recovery mechanisms, we aim to ensure the system can promptly identify and respond to potential faults in edge nodes or the network. Through real-time monitoring of node availability and load conditions, our system will be capable of automatically switching to backup nodes to ensure the continuity and high availability of cache services. This approach will help address unforeseen failures, enhance system stability, and meet the ever-growing user demands.

Acknowledgement: The authors extend their appreciation to the reviewers for their valuable feedback, which enhanced the paper's quality. Gratitude is also expressed to the editor for their guidance during the publication process. Their contributions have greatly contributed to the paper's improvement.

Funding Statement: This article is supported by the Innovation Fund Project of Jiangxi Normal University (YJS2022065) and the Domestic Visiting Program of Jiangxi Normal University.

Author Contributions: The authors confirm contribution to the paper as follows: study conception and design: Yong Ma, Han Zhao, Yunni Xia; data collection: Kunyin Guo, Xu Wang; analysis and interpretation of results: Yong Ma, Han Zhao; draft manuscript preparation: Yong Ma, Han Zhao, Yunni Xia, Xu Wang; manuscript review and editing: Yunni Xia, Xianhua Niu, Dongge Zhu, Yumin Dong. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The datasets employed in this research are openly accessible. The MovieLens dataset is thoroughly documented in [32], and its accessibility is outlined in the same source. As for the Shanghai Telecom dataset, it is comprehensively explicated in [31], and information regarding its acquisition can be found in the corresponding publication. These two datasets have been utilized in strict adherence to their individual usage guidelines and terms of service.

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

References

1. Wang, X., Han, Y., Wang, C., Zhao, Q., Chen, X. et al. (2019). In-Edge AI: Intelligentizing mobile edge computing, caching and communication by federated learning. *IEEE Network, 33(5),* 156–165.
2. Yao, J., Han, T., Ansari, N. (2019). On mobile edge caching. *IEEE Communications Surveys & Tutorials, 21(3),* 2525–2553.
3. Gao, H., Liu, C., Li, Y., Yang, X. (2021). V2VR: Reliable hybrid-network-oriented V2V data transmission and routing considering RSUs and connectivity probability. *IEEE Transactions on Intelligent Transportation Systems, 22(6),* 3533–3546.
4. Huang, M., Jiang, Q., Qu, Q., Rasool, A. (2021). An overlapping community detection approach in ego-splitting networks using symmetric nonnegative matrix factorization. *Symmetry, 13(5),* 869.
5. Reiss-Mirzaei, M., Ghobaei-Arani, M., Esmaeili, L. (2023). A review on the edge caching mechanisms in the mobile edge computing: A social-aware perspective. *Internet of Things, 22,* 100690.
6. Zhang, K., Leng, S., He, Y., Maharjan, S., Zhang, Y. (2018). Cooperative content caching in 5G networks with mobile edge computing. *IEEE Wireless Communications, 25(3),* 80–87.
7. Ndikumana, A., Ullah, S., LeAnh, T., Tran, N. H., Hong, C. S. (2017). Collaborative cache allocation and computation offloading in mobile edge computing. *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, Seoul, Korea (South), IEEE.
8. Tang, Y., Guo, K., Ma, J., Shen, Y., Chi, T. (2019). A smart caching mechanism for mobile multimedia in information centric networking with edge computing. *Future Generation Computer Systems, 91,* 590–600.
9. Wu, Q., Zhao, Y., Fan, Q., Fan, P., Wang, J. et al. (2022). Mobility-aware cooperative caching in vehicular edge computing based on asynchronous federated and deep reinforcement learning. *IEEE Journal of Selected Topics in Signal Processing, 17(1),* 66–81.
10. Wei, H., Luo, H., Sun, Y. (2020). Mobility-aware service caching in mobile edge computing for internet of things. *Sensors, 20(3),* 610.

11. Sadeghi, A., Sheikholeslami, F., Giannakis, G. B. (2017). Optimal and scalable caching for 5G using reinforcement learning of space-time popularities. *IEEE Journal of Selected Topics in Signal Processing, 12(1),* 180–190.
12. Jiang, W., Feng, G., Qin, S., Liang, Y. C. (2019). Learning-based cooperative content caching policy for mobile edge computing. *ICC 2019–2019 IEEE International Conference on Communications (ICC)*, Shanghai, China, IEEE.
13. Ostrowski, K., Małecki, K., Dziurzański, P., Singh, A. K. (2023). Mobility-aware fog computing in dynamic networks with mobile nodes: A survey. *Journal of Network and Computer Applications, 219,* 103724.
14. Zhong, C., Gursoy, M. C., Velipasalar, S. (2020). Deep reinforcement learning-based edge caching in wireless networks. *IEEE Transactions on Cognitive Communications and Networking, 6(1),* 48–61.
15. Song, J., Sheng, M., Quek, T. Q., Xu, C., Wang, X. (2017). Learning-based content caching and sharing for wireless networks. *IEEE Transactions on Communications, 65(10),* 4309–4324.
16. Zhang, F., Ge, J., Wong, C., Li, C., Chen, X. et al. (2019). Online learning offloading framework for heterogeneous mobile edge computing system. *Journal of Parallel and Distributed Computing, 128,* 167–183.
17. Sun, H., Yu, H., Fan, G., Chen, L. (2020). Qos-aware task placement with fault-tolerance in the edge-cloud. *IEEE Access, 8,* 77987–78003.
18. Mitsis, G., Tsiropoulou, E. E., Papavassiliou, S. (2022). Price and risk awareness for data offloading decision-making in edge computing systems. *IEEE Systems Journal, 16(4),* 6546–6557.
19. Zhang, Y., Niu, W., Yan, L. (2024). Irs assisted uav communications against proactive eavesdropping in mobile edge computing networks. *Computer Modeling in Engineering & Sciences, 138(1),* 885–902. https://doi.org/10.32604/cmes.2023.029234
20. Yun, S., Chen, Y. (2023). Intelligent traffic scheduling for mobile edge computing in iot via deep learning. *Computer Modeling in Engineering & Sciences, 134(3),* 1815–1835. https://doi.org/10.32604/cmes.2022.022797
21. 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 Transactions on Vehicular Technology, 73(1),* 348–361.
22. Acheampong, A., Zhang, Y., Xu, X., Kumah, D. A. (2023). A review of the current task offloading algorithms, strategies and approach in edge computing systems. *Computer Modeling in Engineering & Sciences, 134(1),* 35–88. https://doi.org/10.32604/cmes.2022.021394
23. 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 Transactions on Vehicular Technology, 72(5),* 6709–6722.
24. Ghosh, S., Agrawal, D. P. (2021). A high performance hierarchical caching framework for mobile edge computing environments. *2021 IEEE Wireless Communications and Networking Conference (WCNC)*, Nanjing, China, IEEE.
25. Zhao, J., Ma, Y., Xia, Y., Dai, M., Chen, P. et al. (2022). A novel fault-tolerant approach for dynamic redundant path selection service migration in vehicular edge computing. *Applied Sciences, 12(19),* 9987.
26. Zheng, C., Liu, S., Huang, Y., Zhang, W., Yang, L. (2022). Unsupervised recurrent federated learning for edge popularity prediction in privacy-preserving mobile-edge computing networks. *IEEE Internet of Things Journal, 9(23),* 24328–24345.
27. Peng, X., Han, Z., Xie, W., Yu, C., Zhu, P. et al. (2022). Deep reinforcement learning for shared offloading strategy in vehicle edge computing. *IEEE Systems Journal, 17(2),* 2089–2100.
28. Hazarika, B., Singh, K., Li, C. P., Biswas, S. (2022). Multi-agent DRL-based computation offloading in multiple RIS-aided IoV networks. *MILCOM 2022–2022 IEEE Military Communications Conference (MILCOM)*, Rockville, MD, USA, IEEE.

29. Xiao, Y., Wan, K., Xiao, L., Yang, H. (2022). Energy-efficient collaborative inference in mec: A multi-agent reinforcement learning based approach. *2022 8th International Conference on Big Data Computing and Communications (BigCom)*, Xiamen, China, IEEE.
30. AlDurgam, M. M., Duffuaa, S. O. (2010). Optimal maintenance policies for three-states pomdp with quality measurement errors. *2010 IEEE International Conference on Industrial Engineering and Engineering Management*, Macao, China, IEEE.
31. Liu, S., Liu, Y., Ni, L. M., Fan, J., Li, M. (2010). Towards mobility-based clustering. *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Washington DC, USA.
32. Harper, F. M., Konstan, J. A. (2015). The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems, 5(4),* 1–19.
33. Xiao, H., Zhao, J., Pei, Q., Feng, J., Liu, L. et al. (2021). Vehicle selection and resource optimization for federated learning in vehicular edge computing. *IEEE Transactions on Intelligent Transportation Systems, 23(8),* 11073–11087.
34. Banerjee, B., Kulkarni, A., Seetharam, A. (2018). Greedy caching: An optimized content placement strategy for information-centric networks. *Computer Networks, 140,* 78–91.
35. Xu, X., Chen, P., Xia, Y., Long, M., Peng, Q. et al. (2022). MROCO: A novel approach to structured application scheduling with a hybrid vehicular cloud-edge environment. *2022 IEEE International Conference on Services Computing (SCC)*, Barcelona, Spain, IEEE.