Multi-Agent Reinforcement Learning Based File Caching Strategy in Mobile Edge Computing

Yongjian Yang, Kaihao Lou, En Wang*, Wenbin Liu, Jianwen Shang, Xueting Song, Dawei Li, and Jie Wu, *IEEE Fellow*

***Abstract*—Mobile edge computing (MEC) reduces data service latency by pushing data to the network edge. However, due to the dynamic and diverse requests of mobile users, the problem of mobile edge caching is more complex than cloud caching. Therefore, the existing model-based caching strategies cannot be directly used in the mobile edge caching environment. Besides, when taking the cooperative storage relationship between neighbor edge servers into consideration, the caching problem becomes more difficult. To this end, we formulate an mobile edge caching problem to minimize the total latency in mobile edge computing. Firstly, a heuristic caching strategy is proposed to solve the mobile edge caching problem in the single-time-slot scenario. Then, with the consideration of users' mobility and the correlation of files, we propose a caching strategy for the multiple-time-slot scenario based on multi-agent deep reinforcement learning. To address the cold start problem in deep reinforcement learning, we adopt the proposed heuristic caching strategy used in the single-time-slot scenario to further optimize the training results. Extensive experiments on generated data and real-world datasets are conducted to verify that the proposed edge caching strategies can achieve the minimum latency compared with the state-of-the-art strategies.**

***Index Terms*—mobile edge computing, file caching strategy, multi-agent deep reinforcement learning**

I. Introduction

Mobile Edge Computing (MEC) [1] is a novel network architecture that provides computing service at the edge of the mobile network. MEC empowers Mobile Cloud Computing (MCC) by deploying cloud resources, e.g., storage and processing capacity, to the edge within the Radio Access Network (RAN) [2], [3]. In most cases, the edge server needs to provide the corresponding data or files to users according to their requests. However, due to the limited storage capacity of the edge servers, users' requests cannot be well satisfied, which raises the fundamental edge caching problem in MEC [4].

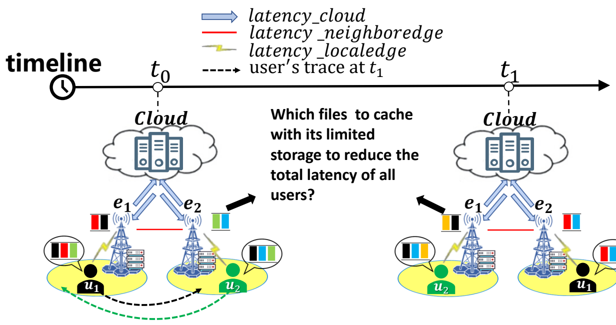


Fig. 1: An example of the mobile edge caching problem.

To provide the edge caching service, the network service providers usually set up the cloud servers in several locations to serve users in a large area, thus the cloud servers are farther away from users in the physical world than the edge servers. Therefore, under the same network conditions (bandwidth and routing), the latency of obtaining files from the cloud server is usually much greater than that of the edge servers. In order to minimize the latency of obtaining files, users should obtain files from the edge server as much as possible. A simple example of the mobile edge caching problem is shown in Fig. 1, it is necessary to design an effiective caching strategy for the edge servers to make efficient use of the limited storage capacity in order to reduce the latency of obtaining files. Some existing works either propose prediction models based on historical patterns [5] and social patterns [6], [7] to predict the popularity of content, or rely on models assuming that the popularity of content is known [8], [9], which are not adaptive enough to the highly dynamic and heterogeneous environment in mobile edge caching.

There are three main challenges in the above edge caching problem, the first one is how to propose a reasonable caching strategy in the single-time-slot scenario based on users' current requests. To deal with the problem, we assume that the requests of users are known at the beginning of the time slot. By considering the impact of the files cached in different edge servers on the total latency, we can estimate the utility of each file. Thus, we propose a heuristic caching strategy with an approximation ratio. The second challenge is how to quantify the impact of cached files in different time slots, users' mobility and the correlation of files, then propose a caching strategy in the multiple-time-slot scenario. In the multiple-time-slot

Manuscript resubmitted January 19, 2022; revised February 22, 2023; accepted April 24, 2023. This work is supported in part by National Key R&D Program of China under Grant Nos. 2022YFB3103700 and 2022YFB3103702, in part by National Natural Science Foundation of China under Grant Nos. 62272193, 62102161, 62072209 and 61972450, in part by CCF-Baidu Open Fund (No. 2021PP15002000), in part by the National Science Foundation (NSF) under Grant CNS 1824440, Grant CNS 1828363, Grant CNS 1757533, Grant CNS 1629746, Grant CNS 1651947, and Grant CNS 1564128, and in part by the Open Project Program of Engineering Research Center of Ecological Big Data, Ministry of Education. (*Corresponding author: En Wang.)

Yongjian Yang, Kaihao Lou, En Wang, Wenbin Liu, Jianwen Shang and Xueting Song are with Department of Computer Science and Technology, Jilin University, Changchun, Jilin, 130012, China. (e-mail: yyj@jlu.edu.cn; loukh20@mails.jlu.edu.cn; wangen@jlu.edu.cn; shangjw20@mails.jlu.edu.cn; songxt20@mails.jlu.edu.cn).

Dawei Li is with the Department of Computer Science, Montclair State University, Montclair, USA. (e-mail: dawei.li@montclair.edu).

Jie Wu is with the Department of Computer and Information Sciences, Temple University, Philadelphia, USA.

scenario, we explore deep reinforcement learning [10], [11] in order to deal with the edge caching problem, where the users' requests are dynamic and diverse. However, traditional deep reinforcement learning usually uses a centralized learning agent, which is not suitable for solving the edge caching problem, because a large number of distributed edge servers will generate an explosive action space. Hence, we further explore multi-agent deep reinforcement learning [12], [13], where each edge server can be regarded as an agent and can cooperate with its neighbor servers to provide caching services. Further considering users' mobility and the correlation of files, we propose an improved multi-agent deep reinforcement learning, which uses predicted results as the observation including the predicted users' mobility and the correlation among different files. However, reinforcement learning usually faces a cold start problem, because of which the algorithm often fails to achieve good training results and each agent cannot determine an effective caching strategy. Therefore, the third challenge of the mobile edge caching problem is how to optimize the performance of the proposed caching strategy in the cold start phase. To deal with the cold start problem, we use the proposed heuristic caching strategy in the single-time-slot scenario to optimize the training results i.e., each agent uses the heuristic caching strategy to determine its own action and collects the feedback from the environment to train its neural networks. After the cold start phase, each agent uses its neural networks to determine its own action for subsequent training.

The main contributions of this paper are summarized as follows:

- For the NP-hard mobile edge caching problem, we first propose a heuristic caching strategy in a single-time-slot scenario and prove the strict tight approximation ratio. This heuristic caching strategy is also used in the cold start phase of the next multiple-time-slot scenario.
- For the multiple-time-slot scenario, we propose a caching strategy based on the improved multi-agent deep reinforcement learning, which is embedded with users' mobility and the correlation of files.
- We conduct extensive simulations based on generated data and three real-world datasets: *roma/taxi* [14], *epfl* [15] and *EUA dataset* [16]. The results show that the proposed caching strategies can achieve the minimum latency when compared to other strategies.

The remainder of this paper is organized as follows. In Section II, we review the related works about mobile edge caching and deep reinforcement learning. The edge caching problem and system model are defined in Section III. We propose a heuristic caching strategy with an approximation ratio in Section IV. The detailed multi-agent deep reinforcement learning based caching strategy is proposed in Section V. In Section VI, we evaluate our caching strategy. We conclude this paper in Section VII.

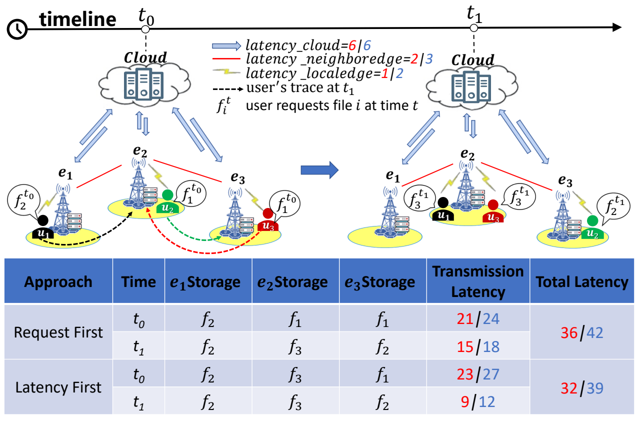
II. Related Work

A. Mobile Edge Caching

As a novel network paradigm, mobile edge caching reduces network traffic and data access service delay by pushing data to the network edge [1]. Most existing edge caching researches propose edge caching strategies by focusing on users [4] or data [17], [18]. However, due to the mobility of users and the diversity of requests, as well as the correlation among files, the environment of mobile edge caching is very complex, and the existing caching strategies cannot solve the problem in mobile edge caching. In order to solve the problem of mobile edge caching in a dynamic heterogeneous environment, recently, caching strategies based on deep reinforcement learning [19]–[21] have been proposed. H. Wu $et\ al.$ [22] propose a novel Edge-oriented Collaborative Caching (ECC) in information-centric networking (ICN) to reduce response latency, server load and bandwidth consumption with the consideration of both content popularity and cache benefit. To reduce the overall energy requirements in mobile edge caching, M. Sarra $et\ al.$ [23] propose an energy-efficient fuzzy caching strategy for edge devices with the consideration of users' mobility, requests, and limited caching size. In [24], S. Rahman $et\ al.$ propose a caching strategy based on deep learning to store content in the edge network in order to provide seamless web content streaming for users. G. Qiao $et\ al.$ [25] propose a cooperative edge caching scheme to jointly optimize the content placement and content delivery in vehicular edge computing networks. Based on the content popularity, vehicle driving paths, and resource availability, the proposed caching scheme can reduce the system cost, as well as the content delivery latency, while improving the content hit ratio. Compared with the above edge caching strategies, our proposed caching strategy based on multi-agent deep reinforcement learning regards the edge servers as agents and considers the cooperation between agents, which is suitable for highly dynamic and heterogeneous environments in mobile edge caching.

B. Deep Reinforcement Learning

There are many applications and research papers which focus on deep reinforcement learning in different fields, such as transportation [26], network [10], [11], etc. In order to better solve the problems in dynamic and heterogeneous environments, multi-agent deep reinforcement learning [12], [13] is proposed. M. K. Sharma $et\ al.$ [27] design a Multi-Agent Reinforcement learning framework to achieve a more effective power control strategy. In [28], Lowe $et\ al.$ propose the multi-agent deep deterministic policy gradient (MADDPG) for cooperative or competitive scenarios. However, when the number of agents is large, the training convergence speed is slow. In order to solve the exponential growth of the interaction caused by the increase of the number of agents, Yang $et\ al.$ propose a method called Mean Field Reinforcement Learning (MFRL) [29], where the interaction between individuals is regarded as the interaction between individuals and collectives. In [30], C. H. Liu $et\ al.$ propose a fully-distributed control solution to navigate a group of UAVs to provide long-term communication service for the ground mobile users, in order to maximize the temporal average coverage, while minimizing the total energy consumptions. There are also researches that combine multi-agent reinforcement learning with emerging fields, such as



Approach	Time	e_1Storage	e_2Storage	e_3Storage	Transmission Latency	Total Latency
Request First	t_0	f_2	f_1	f_1	21/24	36/42
	t_1	f_2	f_3	f_2	15/18	
Latency First	t_0	f_2	f_3	f_1	23/27	32/39
	t_1	f_2	f_3	f_2	9/12	

Fig. 2: Motivating example.

mobile crowdsensing (MCS) [31]–[34]. C. H. Liu *et al.* [35] propose a novel deep learning based framework to enable multiple unmanned aerial vehicles (UAVs) to execute data collection tasks efficiently and cooperatively, while charging the battery from multiple randomly deployed charging stations. Compared with the existing multi-agent deep reinforcement learning, we adopt a heuristic caching strategy to improve the performance of the multi-agent caching strategy during the cold start phase.

TABLE I: Main notations

Symbol	Meaning
E, F, U	the sets of edge servers, files, mobile users.
C	the cloud server.
N_e	the set of neighbor servers of edge server e.
R_u^t	user u's request at time slot t.
$r_{u,f}^t$	whether user u requests file f at time slot t.
$l_{u,s}$	the latency between a user u and a server s.
$l_{e,s}$	the latency between edge server e and another server s (edge server or cloud server).
$\mathcal{L}_t$	the transmission latency to satisfy users' requests at time slot t.
$\mathcal{R}_e^t$	the replacement latency of the edge server e at time slot t.
$x_{u,s,f}^t$	whether a user u's requested file f should be fetched from server s at time slot t.
$y_{e,f}^t$	whether a file f is cached in the edge server e at time slot t.
U_e^t	the set of users whose location is in the area of the edge server e.
$D_e^t(f)$	the utility of file f being cached by an edge server e.
$m_u^t(l_e)$	the probability of user u arrives edge server e's area.
$r_{e,f}^t$	the probability of file f will be requested in the next t time slots for edge server e.

III. Mobile Edge Caching Problem

A. Motivating Example

An example of the mobile edge caching problem is shown in Fig. 2. There are three edge servers $\{e_1, e_2, e_3\}$, each of which can cache one file in a time slot. e_2 is connected with both e_1 and e_3, so e_1 and e_3 are called the neighbor servers of e_2. Each edge server covers a certain area and provides services to users in that area. The latencies of obtaining files from the local edge server, neighbor server, and cloud server are assumed to be 1, 2, 6 (red words) respectively. When the files requested by the user are not cached in its local edge server, the user can fetch the requested files from the neighbor servers or the remote cloud server through the local edge server, since the cloud server caches all the files. There are three different files $\{f_1, f_2, f_3\}$ that may be requested by three users $\{u_1, u_2, u_3\}$, who are moving across the edge server areas. f_i^t means that a user requests f_i at time t. During the two time slots, the users $\{u_1, u_2, u_3\}$ request the files for a total of 6 times. At the beginning of t_0, the edge servers' storage is empty, the users make requests to the edge servers and the edge servers need to decide which files to be cached and obtain the uncached files from the neighbor edge servers or the cloud servers. The cached file remains in the storage until the edge server obtains other files to replace the cached files. It is worth noting that the edge servers may obtain files from the neighbor servers or the cloud server for the users without replacing the cached files in storage.

To minimize the total latency, a simple strategy is **Request First**, where the edge servers cache the requested files of its local users at each time slot. As shown in Fig. 2, the users request $\{f_2, f_1, f_1\}$ at t_0, hence the edge servers decide to cache $\{f_2, f_1, f_1\}$, respectively. In order to cache the above files, the edge servers obtain the files from the cloud server 3 times at t_0 and the latency is $3 \times 6 = 18$. Then the users can fetch the requested files from the local edge servers and the latency is $3 \times 1 = 3$. The total latency at t_0 is $18+3 = 21$. At the beginning of time slot t_1, the users change their locations, and then, they propose their file requests to the local edge servers. In the same way, the latency at t_1 is $6\times2+3\times1 = 15$ and the total latency is $21 + 15 = 36$. Obviously, this caching strategy does not consider the mobility of users and their future requests. Thus, the edge servers need to repeatedly obtain files from the cloud server or the neighbor edge servers to satisfy the requests of users.

In order to further reduce the latency, another caching strategy called **Latency First** is proposed, where the edge servers cache the requested files based on the predicted users' mobility and requests. The detailed caching strategy is shown in the table of Fig. 2. In this case, the edge servers should cache $\{f_2, f_3, f_1\}$ at t_0 instead of $\{f_2, f_1, f_1\}$. The latency of edge servers obtaining the uncached files from the cloud server is $3 \times 6 = 18$. u_1 and u_3 can fetch the requested files from the local edge servers and u_2 needs to fetch the f_1 from the neighbor edge server e_3 through e_2. Therefore, the latency of the user fetching from the edge servers is $2 \times 1 + 1 + 2 = 5$. Thus the latency at t_0 is $18 + 5 = 23$. In the same way, the latency at t_1 is $6 + 3 \times 1 = 9$ and the total latency is $23+9 = 32$. Obviously, the total latency of the second caching strategy is lower than that of the first caching strategy. For another example, when the three latency is set to 2, 3, and 4 (blue words), the performance of **Latency First** is still better than that of **Request First**. Therefore, the caching strategy based on the predicted users' requests may achieve a better performance. However, it is challenging for the edge servers to decide the caching strategies, especially when the users' requests and locations change over time.

B. Problem Definition

The system model of mobile edge caching problem is firstly discussed in this section. There are n edge servers $E = \{e_1, e_2, ..., e_n\}$. The interconnected edge servers are called each others' neighbor edge servers. The neighbor edge servers of the edge server e are denoted as N_e. There are d different files $F = \{f_1, f_2, ..., f_d\}$, each has the same size. Each edge server e has limited storage capacity c_e and the cloud server C has storage capacity to cache all the files F. Each edge server e is located in a specific area and serves users in the area. The edge server in a user's area is called the local edge server of the user. The mobile users $U = \{u_1, u_2, ..., u_m\}$ are moving across different areas. At the beginning of each time slot t, each user u proposes the request $R_u^t = \{r_{u,1}^t, r_{u,2}^t, ..., r_{u,d}^t\}, r_{u,f}^t \in \{0,1\}$ to the local edge server in the current area. $r_{u,f}^t$ represents whether the user u requests the file f at time slot t, if the user u requests f then $r_{u,f}^t = 1$; otherwise, $r_{u,f}^t = 0$. A binary variable $x_{u,s,f}^t \in \{0,1\}, s \in \{E, C\}$ represents whether a user u's requested file f should be obtained from the server s, where s could be an edge server or the cloud server. $y_{e,f}^t \in \{0,1\}, e \in E$ represents whether a file f is cached in the edge server e at time slot t. After a file request arrives, the edge server e will check its storage for the requested file; if the file is not cached, then it will check its neighbor edge servers in N_e. Finally, the edge server has to obtain the requested file from the cloud server if the requested file is not cached in the storage of the edge server or its neighbor edge servers. Generally speaking, the latency of obtaining files from the cloud server is larger than that of edge server. Therefore, in order to minimize the total latency, the edge servers should cache the files that the users request at present and in the future as many as possible.

In the mobile edge caching scene, users may move across different areas and their requests may change. At the beginning of each time slot, the edge servers obtain different requests from the users and decide which files to be cached in their storage. If the files to be cached in this time slot are different from those currently cached in the server's storage, the edge server needs to obtain the uncached files from its neighbor servers or the cloud server to replace the cached file in its storage. The latency that the edge server takes to obtain the uncached files (from neighbor edge servers or the cloud server) and replace the cached files is called the replacement latency. Hence, in this paper, we divide the total latency into two parts: the transmission latency and the replacement latency. The transmission latency includes user-to-edge latency, edge-to-edge latency and edge-to-cloud latency. The latency between a user u and a server s is denoted as $l_{u,s}, u \in U, s \in \{E, C\}$, and the latency between two servers is denoted as $l_{e,s}, e \in E, s \in \{E, C\}$. The transmission latency at t is defined as follows:

$$\mathcal{L}^t = \sum_u^U \sum_f^F x_{u,s,f}^t \times l_{u,s} \times r_{u,f}^t \tag{1}$$

It is worth noting that the user u needs to fetch the requested files from the neighbor edge servers or the cloud server through the local edge server e_u if the requested file is not cached in the storage of the local edge server. The transmission latency of user fetching files should include the latency between the user and the local edge server and the latency of the local edge server obtaining files from the neighbor edge servers or the cloud server. Therefore, $l_{u,s}$ should be calculated as follows:

$$l_{u,s} = \begin{cases} l_{u,e_u}, & \text{if} \quad s = e_u \\ l_{u,e_u} + l_{e_u,s}, & \text{otherwise} \end{cases} \tag{2}$$

The replacement latency of the edge server e at time t is defined as follows:

$$\mathcal{R}_e^t = \sum_f^F \sum_s^{N_e \cup C} \max\{y_{e,f}^{t-1} - y_{e,f}^t, 0\} \times x_{s,f}^{t-1} \times l_{e,s} \tag{3}$$

where $\max\{y_{e,f}^{t-1} - y_{e,f}^t, 0\}$ is used to determine whether the edge server e needs to obtain the file f from other servers. $x_{s,f}^{t-1}$ is the source of the file f obtained by the users in the server e at time slot $t-1$, which is used to determine whether the current edge server e needs to obtain the file f from its neighbor servers or the cloud server.

Based on the above system model, we can define the problem as follows:

Problem (Mobile Edge Caching Problem): At the beginning of each time slot t, each user u may change its location and then propose file request $r_{u,f}^t$ to the local edge server in the current area. Under the limitation of storage capacity, each edge server needs to decide which files to cache in its storage so that the total latency can be minimized, which can be denoted as follows:

$$\text{Minimize} \quad L = \sum_t^T \sum_u^U \sum_f^F x_{u,s,f}^t \times l_{u,s} \times r_{u,f}^t \quad +$$

$$\sum_t^T \sum_e^E \sum_f^F \sum_s^{N_e \cup C} \max\{y_{e,f}^{t-1} - y_{e,f}^t, 0\} \times x_{s,f}^{t-1} \times l_{e,s} \tag{4}$$

$$s.t. \qquad x_{u,s,f}^t \in \{0,1\}, \; s \in E \cup C \tag{5}$$

$$\sum_s^{E \cup C} x_{u,s,f}^t = 1, \forall t \in T \tag{6}$$

$$\sum_f^F y_{e,f}^t \leq c_e, \forall e \in E \tag{7}$$

$$\sum_f^F r_{u,f}^t \geq 1, r_{u,f}^t \in \{0,1\} \tag{8}$$

Constraint (6) ensures that user u's request for file f can be satisfied by one server (including edge server and cloud server). Constraint (7) ensures that the number of files cached in edge server is less than or equal to its storage capacity. Constraint (8) ensures that each user u requests at least one file per time slot.

C. NP-hard Proof

Lemma 1. *The Mobile Edge Caching Problem is NP-hard.*

Algorithm 1 Edge Caching Strategy (ECS)

```
Input: a set of edge servers $E$, a set of files $F$, a set of users $U$ and their requests $R^t$, latency of obtaining files from servers $L$, storage capacity of servers $C_E$
Output: caching lists $\Theta = \{\Theta_e\}, \forall e \in E$
1: Initialize $y^t_{e,f} = 0, \forall e \in E, \forall f \in F$
2: while $E \neq \emptyset$ do
3:     $e, f = \arg\max D^t_e(f)$
4:     $\Theta_e = \Theta_e \bigcup f, y^t_{e,f} = 1$
5:     Update $\Theta$
6:     if $\sum y^t_{e,f} \geq c_e$ then
7:         $E = E - e$
   return $\Theta$
```

Proof. Firstly, we introduce the multicommodity facility location problem. Consider that there is a facility location set $L = \{l_1, l_2, ..., l_m\}$, and a commodity set $C = \{c_1, c_2, ..., c_n\}$. Each user u requests different commodities $C_u \subseteq C$ and the cost for each user to obtain products from facilities in different locations is different. The multicommodity facility location problem is to select M locations from the location set L to build facilities and decide which commodities each facility should produce, so that the total cost of serving all users' requests can be minimized.

Now, we first consider a simplified case of the Mobile Edge Caching Problem. Consider that there are m edge servers and n different files. At time t, each user u requests different files from the local edge server and the latency is different for the user to fetch files from different servers. We can regard the edge servers as the facilities, and regard the files as the commodities. Then the simplified edge caching problem can be regarded as the multicommodity facility location problem, where we do not need to decide where the edge servers should be built, but we need to decide which files to be cached on each edge server. It is obvious that the multicommodity facility location problem is NP-hard. Hence, the edge caching problem is also NP-hard. □

IV. Heuristic Caching Strategy

In this part, a simplified case of mobile edge caching problem is discussed, where there is only one time slot and the mobility of the users is ignored. At the beginning of the time slot, the storage of the edge servers is empty, and the latency that the edge servers need to obtain files from other servers is also ignored.

A. File Utility

First of all, we calculate the utility of a file f being cached by an edge server e, which is denoted as $D_e(f)$. In this paper, the utility of a file is the expectation of how much latency can be reduced if the file is cached by the edge server e, which can be defined as follows:

$$D^t_e(f) = \sum_{s}^{e \cup N_e} \sum_{u}^{U_{e \cup N_e}} r^t_{u,f} \times l^f_u(s) \tag{9}$$

where $U_{e \cup N_e}$ is the set of users whose locations are in the area of edge server e or the area of the neighbor edge server of e. In other words, $U_{e \cup N_e}$ is the set of users who can fetch files from the edge server e. $l^f_u(s)$ is the latency that can be reduced if the edge server e caches f and the user u requests f, which can be calculated as follows:

$$l^f_u(s) = \begin{cases} l^f_{u,j} - l^f_{u,s}, & \text{if } l^f_{u,j} > l^f_{u,s},\ x_{u,j,f} = 1 \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

$j \in \{E, C\}$ is the server, where user u currently obtains the requested file f. $l^f_{u,s}$ represents the latency that user u needs to fetch file f from server s.

B. Heuristic Caching Strategy

In this section, we propose a heuristic caching strategy, which is shown in Algorithm 1. At the beginning of the time slot, the edge servers receive the requests of the users and initialize variable $y^t_{e,f} = 0$ for each edge server and file to ensure the storage of the edge servers is empty. Then, at each iteration, all edge servers will calculate the utility of each file. Then, the edge server e with the highest file utility will cache the file f, which has the highest utility. In other words, the variable $y^t_{e,f}$ will be set to 1. This process is repeated until all edge servers are unable to continue caching files.

C. Approximation Ratio

In this section, the latency of user u obtaining file f from edge server e is denoted as x, the latency of user u obtaining file f from the neighbor servers of edge server e is denoted as y, and the latency of user u obtaining file f from the cloud server C is denoted as z, $0 < x < y < z$.

Theorem 1. *The approximation ratio of Algorithm 1 is $\frac{1+k_1k_2}{2k_1}$, if $k_1 >= 1$ and $k_2 \geq 2k_1$, where $k_1 = \frac{y}{x}$ and $k_2 = \frac{z}{x}$.*

Proof. First of all, we consider a simplified case of the Mobile Edge Caching Problem. Assume that there are three edge servers (a, b, c) located in three different areas and that two different files 0 and 1 may be requested by users. Edge server b is connected to a and c, and three users are located in the three areas of a, b and c, respectively. The edge servers receive file requests 1, 0 and 1 from the users respectively. It is obvious that the optimal caching policies for the edge servers are (1, 0, 1) and the total latency is $3x$. The caching file list of algorithm 1 will be (0, 1, 1) or (1, 1, 0) and the total latency is $x + 2y$. There are two edge servers caching the non optimal files, which we call the *non-optimal edge servers*. In this situation, the approximation ratio is $\frac{x+2y}{3x} = \frac{1+2k_1}{3}$. It is not difficult to find that $\frac{1+k_1k_2}{2k_1} - \frac{1+2k_1}{3} \geq 0$, and the approximation ratio holds. Next, we will prove Theorem 1 under three different cases.

Case 1: The structure of edge servers is the same as the simplified case, and the number of servers is increased to N, the requests of users are $(1, 0, 1, 0, 1, 0, ...)$.

Suppose there are n *non-optimal edge servers*, and we can get:

$$\frac{(N-n)x + ny}{Nx} - \frac{x+2y}{3x} \tag{11}$$

$$= \frac{2k_1(N-n) + 2k_1^2 n - N - k_1k_2N}{2k_1N}. \tag{12}$$

Based on algorithm 1 and the definition of Mobile Edge Caching Problem, the number of *non-optimal edge server* is calculated as follows:

$$n = \lceil \frac{N-2}{4} \rceil \times 2. \tag{13}$$

Then we can get:

$$\frac{2k_1(N-n)+2k_1^2n-N-k_1k_2N}{2k_1N} \leq \frac{k_1(2+N)(1-k_1)-N}{2k_1N} \leq 0. \tag{14}$$

Then the approximation ratio holds.

Case 2: The numbers of requests received by the three edge servers for file 0 and 1 are $(n_0^a, n_1^a, n_0^b, n_1^b, n_0^c, n_1^c)$. We can prove it by contradiction. For a optimal solution (e.g., (1,0,1)), suppose there exists a solution (e.g., (1,1,0)) that breaks the bound limit ($\frac{1+k_1k_2}{2k_1}$), then it can be illustrated as follows:

$$\frac{(n_1^a+n_1^b+n_0^c)x+(n_0^a+n_0^b+n_1^c)y}{(n_1^a+n_0^b+n_1^c)x+(n_0^a+n_1^b+n_0^c)y} > \frac{1+k_1k_2}{2k_1}. \tag{15}$$

Then:

$$(2k_1-k_1k_2-1)n_1^a+(k_1-k_1^2k_2)n_1^b+(2k_1^2-k_1k_2-1)n_1^c+$$
$$(2k_1^2-k_1-k_1^2k_2)n_0^a+(2k_1^2-k_1k_2-1)n_0^b+(k_1-k_1^2k_2)n_0^c > 0. \tag{16}$$

Based on the assumptions of k_1 and k_2 ($k_1 >= 1$ and $k_2 \geq 2k_1$), we can get:

$$2k_1-k_1k_2-1 = k_1-1+k_1(1-k_2) \leq (1-k_2)(k_1-1) < 0, \tag{17}$$
$$k_1-k_1^2k_2 = k_1(1-k_2) < 0, \tag{18}$$
$$2k_1^2-k_1k_2-1 < 2k_1^2-2k_1^2-1 < 0, \tag{19}$$
$$2k_1^2-k_1-k_1^2k_2 = k_1^2(2-k_2)-k_1 < 0, \tag{20}$$
$$k_1-k_1^2k_2 = k_1(1-k_1k_2) < 0, \tag{21}$$

Based on Eq. (17) to (21), we can get

$$(2k_1-k_1k_2-1)n_1^a+(k_1-k_1^2k_2)n_1^b+(2k_1^2-k_1k_2-1)n_1^c$$
$$+(2k_1^2-k_1-k_1^2k_2)n_0^a+(2k_1^2-k_1k_2-1)n_0^b+(k_1-k_1^2k_2)n_0^c < 0. \tag{22}$$

In other words,

$$\frac{(n_1^a+n_1^b+n_0^c)x+(n_0^a+n_0^b+n_1^c)y}{(n_1^a+n_0^b+n_1^c)x+(n_0^a+n_1^b+n_0^c)y} < \frac{1+k_1k_2}{2k_1}, \tag{23}$$

which contradicts Eq. (15). Therefore, the assumption that a solution breaks the bound limit is not tenable, and the approximation ratio holds.

Case 3: Add an edge server d and connect it with server b, which serves one user.

The request obtained by server d could be 0 or 1. If the request is 0, the number of *non-optimal edge server* is 2 or 0. The ratio of heuristic result to optimal solution is $\frac{x+y}{2x} = \frac{1+k_1}{2} < \frac{1+k_1k_2}{2k_1}$ or 1, and the approximation ratio holds. If the request is 1, the number of *non-optimal edge server* is 2. The ratio of the heuristic result to optimal solution is $\frac{x+y}{2x} = \frac{1+k_1}{2} < \frac{1+k_1k_2}{2k_1}$, and the approximation ratio holds.

According to the above three cases, we can prove theorem 1 holds. □

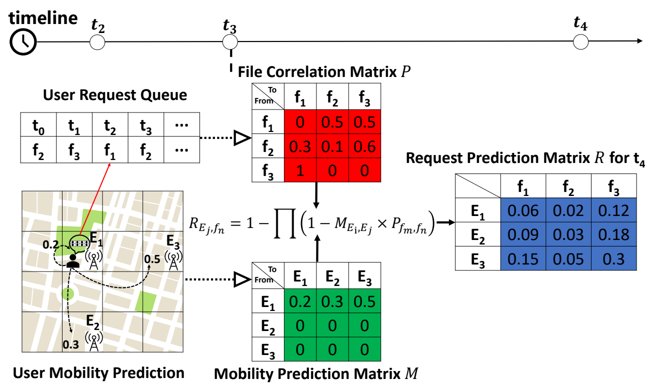

timeline
t_2
t_3
t_4
File Correlation Matrix P
User Request Queue
t_0 t_1 t_2 t_3 ...
f_2 f_3 f_1 f_2 ...
To From f_1 f_2 f_3
f_1 0 0.5 0.5
f_2 0.3 0.1 0.6
f_3 1 0 0
Request Prediction Matrix R for t_4
f_1 f_2 f_3
E_1 0.06 0.02 0.12
E_2 0.09 0.03 0.18
E_3 0.15 0.05 0.3
$R_{E_j,f_n} = 1 - \prod\left(1 - M_{E_i,E_j} \times P_{f_m,f_n}\right)$
0.2
E_1
E_3
0.5
E_2
0.3
To From E_1 E_2 E_3
E_1 0.2 0.3 0.5
E_2 0 0 0
E_3 0 0 0
User Mobility Prediction
Mobility Prediction Matrix M

Fig. 3: Prediction of user's mobility and request.

V. Multi-agent Deep Reinforcement Learning Strategy

In the multiple-time-slot scenario, the edge servers may cache different files in each time slot, and thus there will be replacement latency, which is described in Section III. In this case, the edge caching problem can be considered as a Markov Decision Process (MDP) [36] [37], which is a common model of deep reinforcement learning. Briefly speaking, in MDP, an agent will repeatedly observe the current state s_t of the environment and execute an action a from all available actions. Then, the state of the environment will transfer to s_{t+1} and the agent will get a reward r_t from the environment according to its action. In this paper, the edge servers can be regarded as different agents. At the beginning of each time slot, users may change their locations. Then, they send their requests to the local edge servers. The edge servers need to decide which files should be cached based on the users' requests and their mobility, so that the total latency of users can be minimized. To accelerate the convergence speed, we adopt the Mean Field Reinforcement Learning (MFRL) [29] to address the problem, which is one of the Multi-Agent Reinforcement Learning algorithms. In MFRL, the interaction between agents is regarded as the interaction between individuals and collectives, which is suitable to deal with the problem in multi-agent deep reinforcement learning with a large number of agents.

A. Problem Formulation

1) State and Observation: First of all, we will discuss the state space of mobile edge caching problem. The state of an edge server e at time slot t is denoted as $\mathbb{S}_e^t = \{\mathbb{Y}_e^t, \mathbb{R}_e^t, \mathbb{N}_e\}$, where $\mathbb{Y}_e^t = \{y_{e,f}^t\}, \forall f \in F$ represents whether a file f is cached in the edge server e at the beginning of the time slot t. $\mathbb{R}_e^t$ is the requests, which the edge server e receives at the beginning of the time slot t and $\mathbb{N}_e$ is the neighbor edge servers of edge server e. In the real world, sharing states between edge servers does not require much bandwidth resources, thus the overhead is ignored in this paper. The observation of an edge server e includes its own state and its neighbor servers' states, which is denoted as follows:

$$\mathcal{O}_e^t = \{\mathbb{S}_e^t, \{\mathbb{S}_i^t\}_{i \in N_e}, M_U^{t+1}, P_{e,F}\} \tag{24}$$

where $M_U^{t+1} = \{m_u^{t+1}(l_e)\}, \forall u \in U, m_u^{t+1}(l_e) \in [0,1]$ and $m_u^{t+1}(l_e)$ is the prediction of the user u arriving at the current edge server e's area l_e in the next time slot $(t+1)$. $P_{e,F} = \{p_{e,f}^{t+1}\}, \forall f \in F, p_{e,f} \in [0,1]$ is the predicted probability of file f being requested at edge server e. The details will be described in the next two parts.

- **User Mobility Prediction:** In the mobile edge caching scene, users are moving between the areas, which may have a huge impact on the total latency. In this paper, we assume that we have collected the users' historical trajectories. Thus, we can map the trajectories into a square area [38]. Then the area can be divided into h grids and the users' historical trajectories can be converted into a series of grid coordinates. Based on the processed users' trajectory data, in this paper, we adopt the semi-markov model [39], [40] to predict the users' mobility, and a simple case is shown in the lower part of Fig. 3. One of the most important equations of semi-markov, $Z(\cdot)$ is defined by Eq. (25), which is shown as follows:

$$\begin{aligned} Z_u(l_i, l_j, T) =& P(S_u^{n+1} = l_j, t_u^{n+1} - t_u^n \le T | S_u^0, ..., S_u^n; \\ & t_u^0, ..., t_u^n) \\ =& P(S_u^{n+1} = l_j, t_u^{n+1} - t_u^n \le T | S_u^n = l_i) \end{aligned} \tag{25}$$

where $Z_u(l_i, l_j, T)$ is the probability that the user u will move to the grid (location) l_j in the next move from his/her current grid l_i. S_u^k represents the user u's k-th location in his/her movement and its corresponding arrival time is denoted as t_u^k. The grid that the customer will enter in the next time unit is related to his/her current grid, which can be obtained from the customer's historical trace records. Then, we can define another key equation $Q(\cdot)$, denoted by Eq. (26).

$$Q_u(l_i, l_j, T) = \begin{cases} \Sigma_{k=1}^{h} \Sigma_{t=1}^{T} (Z_u(l_i, l_k, t) - Z_u(l_i, l_k, t-1)) \cdot \\ Q_u(l_k, l_j, T-t), \quad i \neq j \\ 1 - \Sigma_{k=1, k\neq i}^{h} Z_u(l_i, l_k, T) + \\ \Sigma_{k=1,k\neq i}^{h} \Sigma_{t=1}^{T} (Z_u(l_i, l_k, t) - Z_u(l_i, l_k, t-1)) \cdot \\ Q_u(l_k, l_i, T-t), \quad i = j \end{cases} \tag{26}$$

$Q_u(\cdot)$ denotes the probability that a user u arrives l_j from l_i after T time slots. Since the user cannot move from one grid to another when $T = 0$, it is obvious that $Q_u(l_i, l_i, 0) = 1$ and $Q_u(l_i, l_j, 0) = 0$ $(i \neq j)$. Next, we calculate the probability of a user u arriving at any edge server e's area l_e at time slot t, which is shown as follows:

$$m_u^t(l_e) = 1 - \prod_{i=0}^{h} (1 - Q_u(l_i, l_e, t)) \tag{27}$$

- **File Request Prediction:** The other important factor affecting the caching strategy of edge servers is the future requests of the users. If the edge server can accurately predict the future requests of the users, then it can make better caching strategies. To predict the users' future requests, we consider the correlation between files. In this paper, file correlation is defined as the probability that a file f' will be requested in the next time slot after another file f is requested, which is widespread in reality. Taking online video service as an example, after the user watches the first video clip f, if the user is interested in the video content, then he is likely to request the next video clip f' in the next time slot. The correlation between files can be obtained by the history of users' requests, which is assumed to be collected in this paper.

Similar to the prediction of users' mobility, we can count how many times users requesting the file f, which is denoted as n_f. The number of times that file f' is requested in the next time slot after the file f is requested can also be counted as $n_{f,f'}$. Then, in this paper, the file correlation is calculated by Eq. (28), which is shown as follows:

$$p_{f,f'} = \frac{n_{f,f'}}{n_f} \tag{28}$$

After the file correlation is calculated, we will predict the requests of different files on edge servers. Since the probability of users arriving at different edge servers areas is different, the probability of each file being requested in the future is also different for different edge servers. Therefore, we need to predict the probability that each file f is requested in the future for each server e. Based on the prediction of users' mobility and file correlation, the probability of a file f being requested in the next t time slots for the edge server e is calculated as follows:

$$r_{e,f}^t = 1 - \prod_{u}^{U} [1 - m_u^t(l_e) \times p_{f',f}] \tag{29}$$

where f' is the file that is requested in this time slot by the user u. A simple case is shown in the upper part of Fig. 3. Obviously, the probability that file f is requested in the next t time slots can be achieved by calculating the probability that at least one user requests the file f in the next t time slots.

2) Action Space: At the beginning of each time slot, each edge server e needs to decide which files to be cached in its storage to minimize the total latency of all users. The action of an edge server e is denoted as $a_e^t = \{a_{e,f}^t\}, \forall f \in F, a_{e,f}^t \in \{0,1\}$. If e decides to cache file f in this time slot, $a_{e,f}^t = 1$, else $a_{e,f}^t = 0$. Obviously, $\sum a_{e,f}^t \le c_e, \forall f \in F$.

3) Reward Function: After each edge server has taken an action, the current state of each edge server and the whole environment will change and the edge servers will obtain the rewards based on their actions and the requests of users. In this paper, the reward of edge server e at time slot t is defined as the negative total latency of the users in the edge server e's area, including transmission latency and replacement latency. And the reward function of the edge server e is defined as follows:

$$\begin{aligned} r_e^t = -(\alpha \sum_{u}^{U_e} \sum_{r}^{R_u} x_{u,s,f}^t \times l_{u,s} \times r_{u,f}^t + \\ \beta \sum_{f}^{F} \sum_{s}^{N_e \cup C} \max\{a_{e,f}^{t-1} - a_{e,f}^t, 0\} \times x_{s,f}^{t-1} \times l_{e,s}) \end{aligned} \tag{30}$$

where the first part is the transmission latency of users in this edge server area and the second part is the replacement latency of edge server e's action. α and β are the weights of importance of the latency.

4) Problem Formulation: We have defined the entire mobile edge caching scene. At the beginning of each time slot, the edge servers will obtain the requests from the users after the users move. Then the edge servers will take the users'

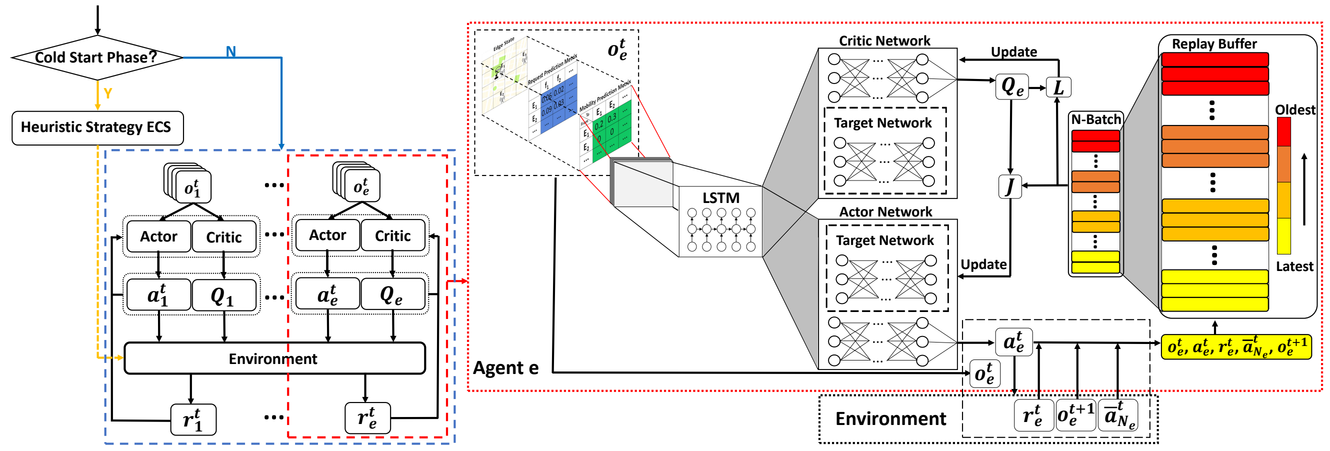


Fig. 4: Mobile Edge Caching Framework.

requests and the prediction of users' mobility and requests as the observation. Each edge server decides what action to take to reduce the total latency, based on its observation. For each edge server e, the problem can be formulated as:

$$V_e^t(\mathcal{O}_e^t) = \max_{a_e^t} \left[r_e^t(\mathcal{O}_e^t, a_e^t) + \gamma V_e^{t+1}(\mathcal{O}_e^{t+1}) \right] \tag{31}$$

where $V_e^t(\mathcal{O}_e^t)$ is the expectation of total latency for edge server e under the observation $\mathcal{O}_e^t$. It is worth noting that $V_e^t(\mathcal{O}_e^t)$ is negative, so the total latency can be minimized by maximizing $V_e^t(\mathcal{O}_e^t)$.

For each edge server e, the optimal caching strategy π_e is given by:

$$\pi_e = \arg\max_{a_e^t} \left[r_e^t(\mathcal{O}_e^t, a_e^t) + \gamma V_e^{t+1}(\mathcal{O}_e^{t+1}) \right] \tag{32}$$

B. Caching Strategy

1) Taking Actions Based on Observations: At the beginning of a time slot, each edge server e will obtain the observation of its state and its neighbor servers' states. Then, based on the observations, the edge servers will decide which files to be cached as their actions and execute the actions.

2) Storing Transitions: After the edge servers have executed their actions, the users will try to fetch files from the current edge server to satisfy their requests. The files could be fetched from the current edge server and the neighbor edge servers, or the cloud server. At the end of the time slot, users may change their locations and their requests, and each edge server would obtain a reward for its action in this time slot. Hence, the edge server e could obtain a tuple consisting of the current observation, action, reward, average action of neighbor servers, and next observation, which is denoted as $(\mathcal{O}_e^t, a_e^t, r_e^t, \overline{a}_{N_e}^t, \mathcal{O}_e^{t+1})$. The tuples at different time slots should be stored in the edge server's replay buffer for training. The average action of neighbor servers is defined as follows:

$$\overline{a}_{N_e}^t = \frac{1}{|N_e|} \sum_{s}^{N_e} a_s^t \tag{33}$$

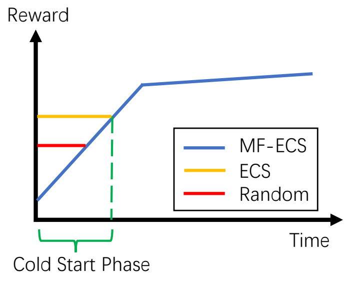


Fig. 5: An example of cold start phase.

3) Training Neural Networks: There are two main neural networks in our MFRL model: the actor network π^e and the critic network Q^e. The detailed neural network structure is shown in Fig. 4. It should be noted that including the historical requests into the observation will also greatly increase the input space, making it difficult for training. To solve this problem, we adopt long short term memory (LSTM), which is an advanced learning model that is widely used for time series processing, to capture the hidden content request patterns. Specifically, the LSTM layer is deployed at the last layer before the actor and critic networks, so that the feature representations of the past time periods can be integrated together for the action decision at the current time period. The actor network π^e is used to determine the action for the agent in this time slot. At the beginning of a time slot, each edge server inputs its observation to its actor network and gets the action for this time slot. The critic network Q^e is used to estimate the reward of different actions. For example, after an edge server e has executed the action, the next time this edge server obtains the same observation, if this edge server executes a different action and gets a higher reward, then the actor network should increase the probability of taking the new action. Hence, we can propose the method to update two neural networks.

For the critic network, each edge server e should sample N batches from its replay buffer, and update the critic network by minimizing the loss function. The loss function is defined as Eq. (34).

Algorithm 2 Edge Caching Strategy (MF-ECS)

```
Input: a set of edge servers $E$, a set of users $U$, and a set of files $F$
Output: target actor and critic networks
1:  Initialize discount factor $\gamma$, update rate $\tau$;
2:  for edge server $e \in E$ do
3:      Randomly initialize critic network $Q^e$ and actor network $\pi^e$;
4:      Initialize target critic network $Q'^e = Q^e$ and target actor network $\pi'^e = \pi^e$;
5:      Initialize the replay buffer $B_e$;
6:  for episode = $1, 2, ..., N$ do
7:      Initialize environment;
8:      for epoch t = $1, 2, ..., T$ do
9:          for edge server $e \in E$ do
10:             if episode $< N'$ then
11:                 Get observation $\mathcal{O}_e^t$, and select action $a_e^t$ by heuristic caching strategy ECS from Algorithm 1
12:             else
13:                 Get observation $\mathcal{O}_e^t$, and select action $a_e^t = \pi^e(\mathcal{O}_e^t)$ by actor network $\pi^e$
14:         Execute each edge server's action $a_e^t$, get observation $\mathcal{O}_e^{t+1}$ and reward $r_e^t$
15:         for edge server $e \in E$ do
16:             Calculate $\overline{a}_{N_e}^t = \frac{1}{|N_e|}\sum_s^{N_e} a_s^t$
17:             Store transition $(\mathcal{O}_e^t, a_e^t, r_e^t, \overline{a}_{N_e}^t, \mathcal{O}_e^{t+1})$ into $B_e$
18:             Randomly sample $N$ batches from $B_e$
19:             Update critic network and actor network by algorithm 3
```

Algorithm 3 Update the Critic and Actor Networks

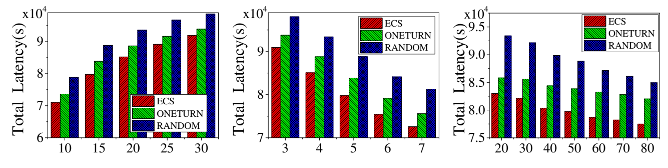
```
1: $y = r_e^t(\mathcal{O}_e^t, a_e^t) + \gamma Q'^e(\mathcal{O}_e^{t+1}, a_e^{t+1}, \overline{a}_{N_e}^{t+1})$
2: $L(Q^e) = \frac{1}{N}\sum(Q^e(\mathcal{O}_e^t, a_e^t, \overline{a}_{N_e}^t) - y)^2$
3: Update the critic network by minimizing $L(Q^e)$
4: Update the actor network by using gradients:
5: $\nabla_{\phi_e} J(\phi_e) \approx \frac{1}{N}\sum \nabla_{\phi_e} \log \pi_{\phi_e}(\mathcal{O}_e^t)$
6:         $Q^e(\mathcal{O}_e^t, a_e, \overline{a}_{N_e})|_{a=\pi_{\phi_e}(\mathcal{O}_e^t)}$
7: Update the weights of corresponding target networks by:
8: $w_{Q'^e} = \tau w_{Q^e} + (1-\tau) w_{Q'^e}$
9: $w_{\pi'^e} = \tau w_{\pi^e} + (1-\tau) w_{\pi'^e}$
```

$$\begin{aligned} L(Q^e) &= \mathbb{E}[(Q^e(\mathcal{O}_e^t, a_e^t, \overline{a}_{N_e}^t) - y)^2] \\ y &= r_e^t(\mathcal{O}_e^t, a_e^t) + \gamma Q'^e(\mathcal{O}_e^{t+1}, a_e^{t+1}, \overline{a}_{N_e}^{t+1}) \end{aligned} \tag{34}$$

After we have trained the critic network, we need to train the actor network, which is shown in Eq. (35).

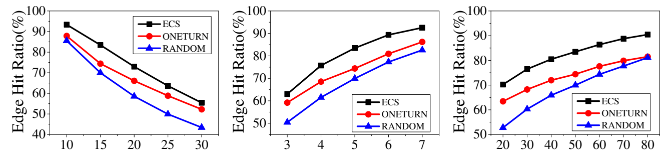
$$\begin{aligned} \nabla_{\phi_e} J(\phi_e) &= \nabla_{\phi_e} \log \pi_{\phi_e}(\mathcal{O}_e^t) \\ &\quad Q^e(\mathcal{O}_e^t, a_e, \overline{a}_{N_e})|_{a=\pi_{\phi_e}(\mathcal{O}_e^t)} \end{aligned} \tag{35}$$

It is worth noting that at the beginning of the training of reinforcement learning, which is called the cold start phase, there is not enough feedback collected from the environment, and the initialization of neural network is random, the edge caching strategy in the cold start phase often leads to bad performance. An example of cold start phase is shown in Fig. 5, with the continuous iteration, the reward of strategy MF-ECS is increasing, which is based on Multi-Agent Reinforcement Learning. However, in the cold start phase, the reward of MF-ECS is less than ECS, or even less than Random strategy. Therefore, in order to reduce the total latency of the cold start phase, we adopt the results of the proposed heuristic strategy as agents' actions and collect feedback from the environment for subsequent training.



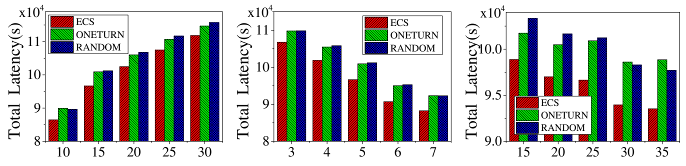
(a) File Type Number (b) Storage Capacity (c) Topology Complexity

Fig. 6: Total latency on *roma/taxi* in single-time-slot scenario.



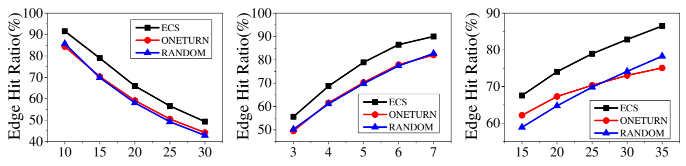
(a) File Type Number (b) Storage Capacity (c) Topology Complexity

Fig. 7: Hit ratio on *roma/taxi* in single-time-slot scenario.



(a) File Type Number (b) Storage Capacity (c) Topology Complexity

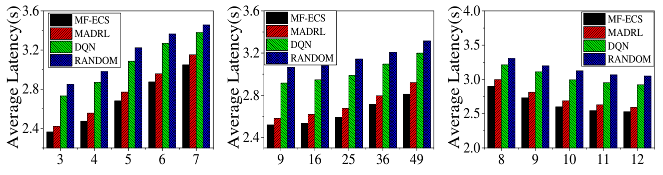
Fig. 8: Total latency on *EUA* in single-time-slot scenario.



(a) File Type Number (b) Storage Capacity (c) Topology Complexity

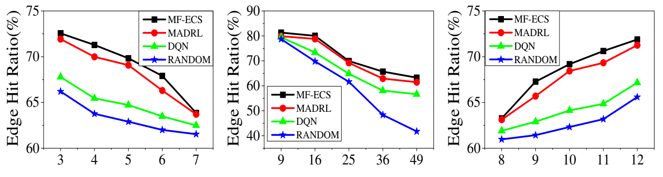
Fig. 9: Hit ratio on *EUA* in single-time-slot scenario.

4) Algorithm Descriptions: The detailed edge caching strategy is shown in algorithm 2. At the beginning of the training process, we initialize the parameters, environment and neural networks for the training (lines 1-5). Each edge server e is regarded as an agent and obtains its observation at the beginning of each epoch (time slot). In the initial N' episodes (cold start phase), the agents decide their actions by using the heuristic strategy ECS (line 11) and the neural networks are updated according to the collected tuples. After cold start phase, each agent decides its action by feeding its actor network with its observation (lines 6-13). After all agents have decided their actions, the caching list of agents will be updated and the users will try to fetch files from the local edge servers. The rewards are calculated based on the transmission latency and replacement latency. The above information is stored as tuples by each agent into its replay buffer. Next, the neural networks are updated by sampling tuples from the replay buffer (lines 14-19).



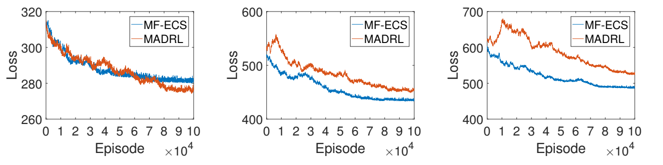
(a) Request Length (b) Edge Server Number (c) Storage Capacity

Fig. 10: Average latency on *roma/taxi* in multiple-time-slot scenario.



(a) Request Length (b) Edge Server Number (c) Storage Capacity

Fig. 11: Hit ratio on *roma/taxi* in multiple-time-slot scenario.

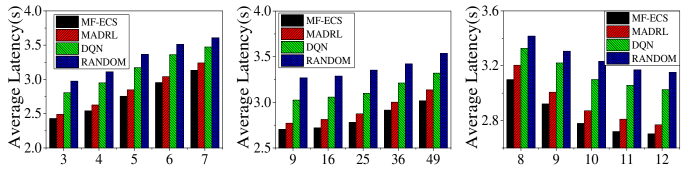


(a) roma/taxi (b) epfl (c) EUA

Fig. 12: Average Training Loss

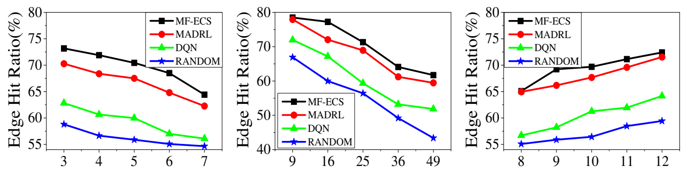
TABLE II: Approximation Ratio on *roma/taxi*.

Edge Servers	6	7	8	9	10
ECS / Optimal	1.0094	1.0039	1.0029	1.0027	1.0073
$\frac{1+k_1k_2}{2k_1}$	2.25	2.25	2.25	2.25	2.25



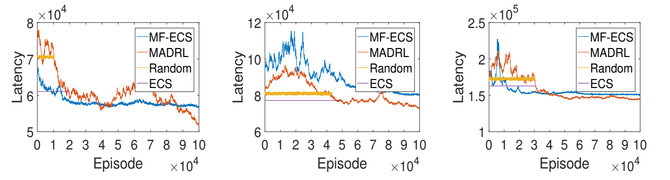
(a) Request Length (b) Edge Server Number (c) Storage Capacity

Fig. 13: Average latency on *EUA* in multiple-time-slot scenario.



(a) Request Length (b) Edge Server Number (c) Storage Capacity

Fig. 14: Hit ratio on *EUA* in multiple-time-slot scenario.



(a) roma/taxi (b) epfl (c) EUA

Fig. 15: Average Training Latency

TABLE III: Approximation Ratio on *EUA*.

Edge Servers	6	7	8	9	10
ECS / Optimal	1.0052	1.0046	1.0057	1.0047	1.0027
$\frac{1+k_1k_2}{2k_1}$	2.25	2.25	2.25	2.25	2.25

VI. Performance Evaluation

A. The Simulation Traces and Settings

In this paper, three real-world datasets are adopted:

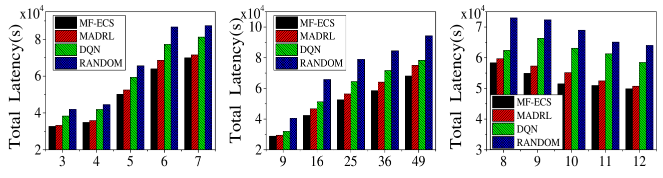
- *roma/taxi* trace set [14]: In *roma/taxi* trace set, 320 taxi drivers that work in the center of Rome city are included. The trajectories in this dataset are collected every 7 seconds and sent to a central server, which represent the positions of those taxi drivers. It includes more than 15,000 trajectory information from more than 150 users.
- *epfl* trace set [15]: In the *epfl* trace set, there are about 500 taxis' GPS coordinates, which are collected over 30 days in the San Francisco Bay Area. Each taxi is equipped with a GPS receiver and sends a location-update to a central server. It includes more than 35000 trajectory information from more than 300 users.
- *EUA dataset* [16]: This dataset is collected by the Australian Communications and Media Authority (ACMA) including the geographical location of all cellular base stations in Australia, which are used as the locations of edge servers. The coverage of each edge server is randomly set within a range of 450-750 meters. It includes more than 90,000 base stations and 100,000 user location information.

For the users' requests, because we cannot get the real request dataset, we adopt the generated data. In order to highlight the impact of file correlation on the results, the users' requests data is generated based on targeted settings.

We implement the MF-ECS using tensorflow [41], which runs on a server with RTX 2080 Ti GPU cards, Intel Core-i9 3.7 GHz CPU cards and 64GB memory. The α and β in Eq. (30) are set to 1. The detailed size of actor and critic networks is $512 \times 256 \times 128$ (not including the input layer and output layer, which are set according to the state-action space). The edge servers area is divided into a grid-shape. To simplify the problem, the neighbor edge servers of an edge server are the edge servers in the upper, lower, left and right regions. The learning rate and the update rate τ are set as 0.001, discount factor $\gamma = 0.9$. The batch size N is set to 1024. The latency of obtaining files from the local edge server is set to 1, while the latency of obtaining files from neighbor edge server and cloud server is set to 2 and 4.

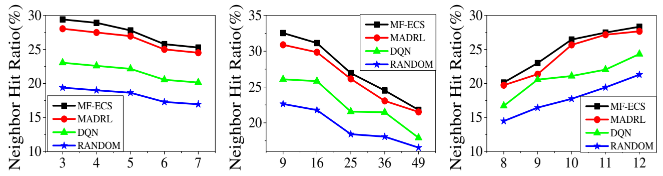
B. Baseline Methods

- *MADRL* [20]: Multi-agent deep reinforcement learning based on A2C.
- *Deep Q-Learning (DQN)*: Each edge server determines its caching policy by using Deep Q-Learning [42].
- *ONETURN*: Each edge server caches the files with maximum file utilities.



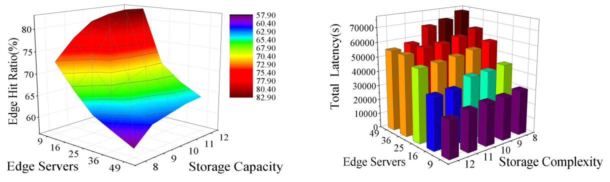
(a) Request Length (b) Edge Server Number (c) Storage Capacity

Fig. 16: Total latency on *roma/taxi* in multiple-time-slot scenario.



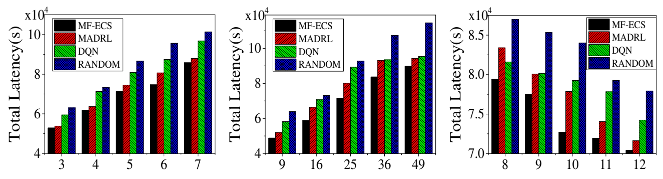
(a) Request Length (b) Edge Server Number (c) Storage Capacity

Fig. 17: Neighbor hit ratio on *roma/taxi* in multiple-time-slot scenario.



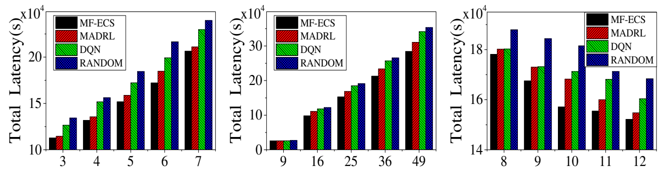
(a) Hit ratio on *roma/taxi* (b) Total latency on *roma/taxi*

Fig. 18: Hit ratio and total latency on *roma/taxi* in multiple-time-slot scenario.



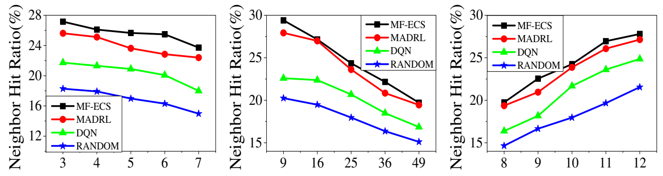
(a) Request Length (b) Edge Server Number (c) Storage Capacity

Fig. 19: Total latency on *epfl* in multiple-time-slot scenario.



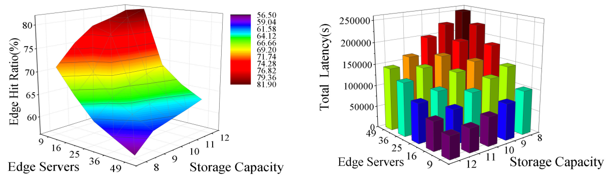
(a) Request Length (b) Edge Server Number (c) Storage Capacity

Fig. 20: Total latency on *EUA* in multiple-time-slot scenario.



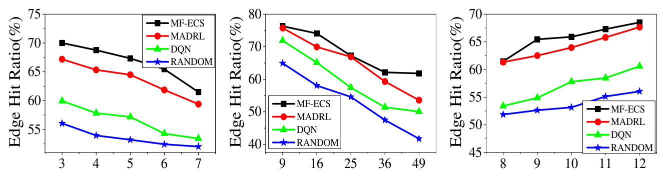
(a) Request Length (b) Edge Server Number (c) Storage Capacity

Fig. 21: Neighbor hit ratio on *EUA* in multiple-time-slot scenario.



(a) Hit ratio on *EUA* (b) Total latency on *EUA*

Fig. 22: Hit ratio and total latency on *EUA* in multiple-time-slot scenario.



(a) Request Length (b) Edge Server Number (c) Storage Capacity

Fig. 23: Hit Ratio on *epfl* in multiple-time-slot scenario.

- *RANDOM*: Each edge server randomly selects the files to be cached in its storage.
- *MF-NMP*: The MF-ECS strategy without users' mobility prediction.
- *MF-NRP*: The MF-ECS strategy without users' request prediction.

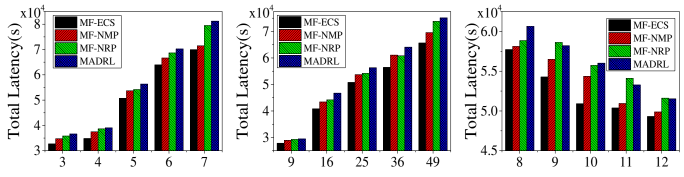
C. Metrics

We adopt five metrics in our simulations:

- *Total Latency*: The total latency for all users' requests to be satisfied in a time period.
- *Average Latency*: Average latency for a user's request to be satisfied.
- *Edge Hit Ratio*: The probability of a user's requests can be satisfied by the current edge server and its neighbor edge servers.
- *Neighbor Hit Ratio*: The probability of a user's requests can be satisfied by the current edge server's neighbor edge servers.
- *D-value of Latency*: The difference value of total latency between MF-ECS and RANDOM.
- *D-value of Hit Ratio*: The difference value of hit ratio between MF-ECS and RANDOM.

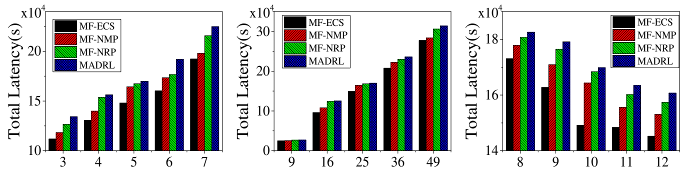
D. Heuristic Caching Strategy

1) Performances on Total Latency and Hit Ratio: We show the total latency and the hit ratio of the proposed heuristic caching strategy in the single-time-slot scenario, and the results are shown in Fig. 6 - Fig. 9. It is not difficult to find that our proposed heuristic caching strategy ECS can achieve lower latency and higher hit ratio compared with other caching strategies. With the increase of the storage capacity of the edge servers, the edge servers can cache more files, so the hit ratio of the edge servers will be improved. At the same time, the total latency will also decrease because the users' requests can be satisfied by the edge servers. When the topology complexity of an edge server increases, the number of neighbor edge servers of the edge server will increase, and users' requests



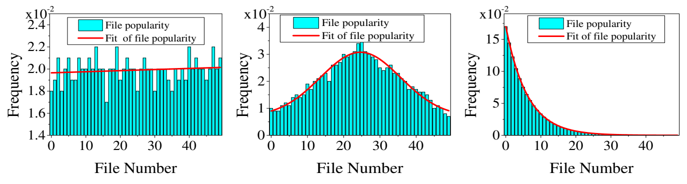
(a) Request Length (b) Edge Server Number (c) Storage Capacity

Fig. 24: Total latency on *roma/taxi*.



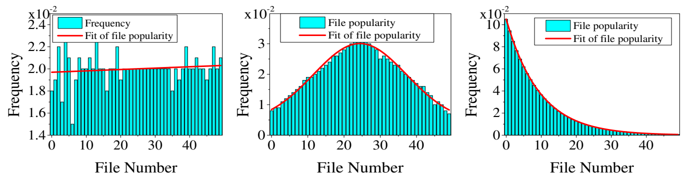
(a) Request Length (b) Edge Server Number (c) Storage Capacity

Fig. 28: Total latency on *EUA*.



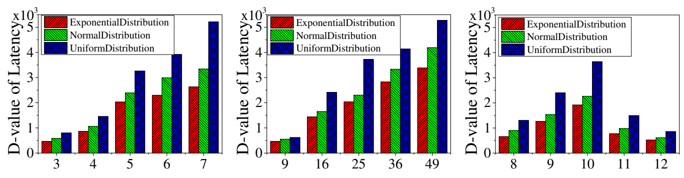
(a) Uniform Distribution (b) Normal Distribution (c) Exponential Distribution

Fig. 25: Different distributions of file popularity on *roma/taxi*.



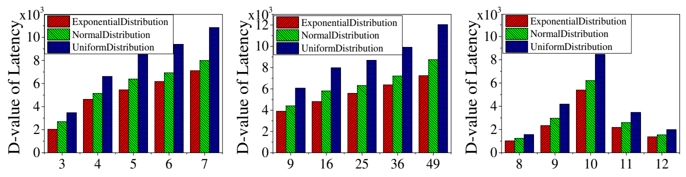
(a) Uniform Distribution (b) Normal Distribution (c) Exponential Distribution

Fig. 29: Different distributions of file popularity on *EUA*.



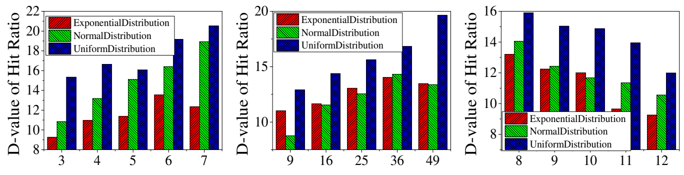
(a) Request Length (b) Edge Server Number (c) Storage Capacity

Fig. 26: D-value of latency on *roma/taxi*.



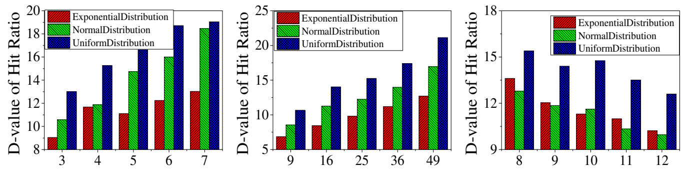
(a) Request Length (b) Edge Server Number (c) Storage Capacity

Fig. 30: D-value of latency on *EUA*.



(a) Request Length (b) Edge Server Number (c) Storage Capacity

Fig. 27: D-value of hit ratio on *roma/taxi*.



(a) Request Length (b) Edge Server Number (c) Storage Capacity

Fig. 31: D-value of hit ratio on *EUA*.

can be better satisfied by the neighbor edge servers; thus, the total latency is reduced.

2) Performances on Approximation Ratio: Next, we conduct simulations to verify the correctness the approximation ratio, which are shown in Table II and Table III. We can find that with different numbers of the edge servers, the ratios of the heuristic results to the optimal results are always smaller than the approximation ratio proved in this paper, which can further prove the correctness and validity of the approximation ratio in this paper.

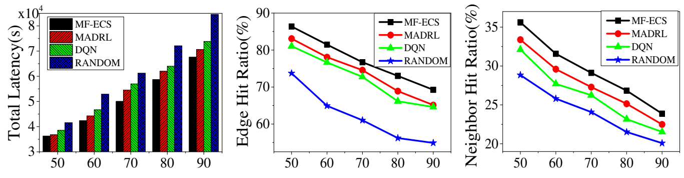
E. Multi-agent Caching Strategy

1) Performances on Average Latency: We first evaluate the performances of different caching strategies on average latency; the results for these simulations are shown in Fig. 10 and Fig. 13. The user request length indicates the maximum number of files that the user can request in a time slot. We can find that with the increase of user request length, the average latency increases, and the hit ratio decreases. The reason for this phenomenon is that the diversity of requests caused by the increase of the length of user requests leads to the decrease of hit ratio of edge servers, and improves the average latency.

When the storage capacity of the edge servers increases, since the edge servers can cache more files at each time slot, the probability that the user can be directly served by the local edge server and its neighbor edge servers will increase, and thus the average latency can be reduced. In this case, the average latency of our caching strategy is about 25% lower than that of RANDOM. In addition, the average training loss and latency are shown in Fig. 12 and Fig. 15. Compared with the Actor-Critic based MADRL method, the proposed MF-ECS strategy converges faster in average training loss.

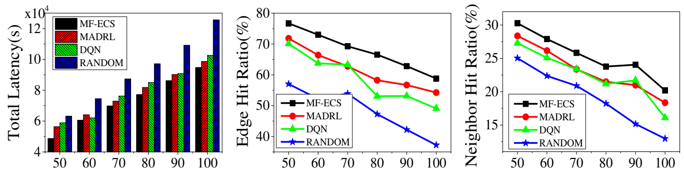
2) Performances on Edge Hit Ratio: After we test the performances of different strategies on average latency, we test the performances of these strategies on edge hit ratio, the results of which are shown in Fig. 11, Fig. 14 and Fig. 19. The edge hit ratio of these strategies can be ranked as MF-ECS $>$ MADRL $>$ DQN $>$ RANDOM. The RANDOM strategy cannot specifically adjust the files cached by the edge servers according to the users' requests. Therefore, the edge servers' hit ratio of RANDOM strategy is the lowest. Compared with other strategies, MF-ECS considers the interaction between agents, and also considers the impact of users' mobility and the correlation between different files, so it can achieve the highest hit ratio under different conditions.

3) Performances on Total Latency and Neighbor Hit Ratio: Next, we conduct the simulations to test performances of different caching strategies on total latency, the results of



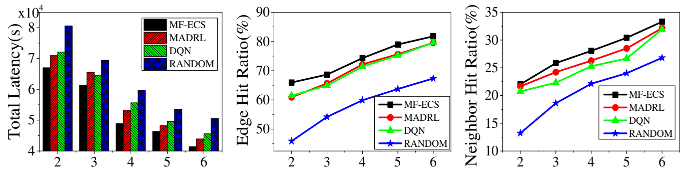
(a) User number (b) User number (c) User number

Fig. 32: Performance when changing the number of users on *roma/taxi*.



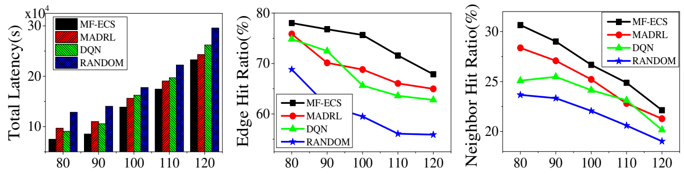
(a) File type number (b) File type number (c) File type number

Fig. 33: Performance when changing the type of files on *roma/taxi*.



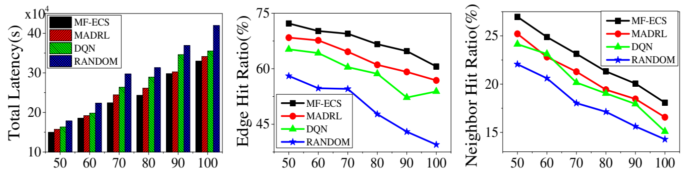
(a) Neighbor number (b) Neighbor number (c) Neighbor number

Fig. 34: Performance when changing the number of neighbor on *roma/taxi*.



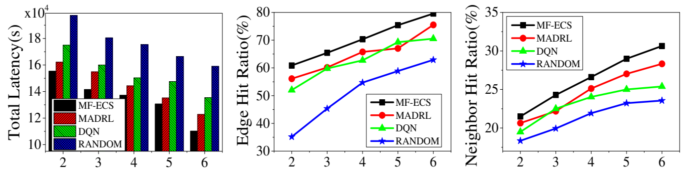
(a) User number (b) User number (c) User number

Fig. 35: Performance when changing the number of users on *EUA*.



(a) File type number (b) File type number (c) File type number

Fig. 36: Performance when changing the type of files on *EUA*.



(a) Neighbor number (b) Neighbor number (c) Neighbor number

Fig. 37: Performance when changing the number of neighbor on *EUA*.

which are shown in Fig. 16, Fig. 20 and Fig. 23. Similar to the previous results, the proposed MF-ECS strategy can achieve the lowest total latency under several different conditions. Besides, the performances on neighbor hit ratio of different strategies are shown in Fig. 17 and Fig. 21. Obviously, the proposed caching strategy MF-ECS can achieve the highest neighbor hit ratio. It is worth noting that there is a negative correlation between hit ratio and latency. This is reasonable since, due to the increase in the hit ratio, users' requests can be better satisfied by the edge servers, which can reduce the probability of users obtaining files from the cloud server, and thus the latency is reduced.

We also illustrate the results of changing the number of edge servers and storage capacity, which are shown in Fig. 18 and Fig. 22. Similar to the previous results, as the server's storage capacity increases, the servers' hit ratio will increase, and the total latency will decrease. When the number of edge servers increases, due to the increase of the number of users, the servers' hit ratio decreases due to the diverse requests of users, and some requested files need to be obtained from the cloud server, so the total latency will also increase.

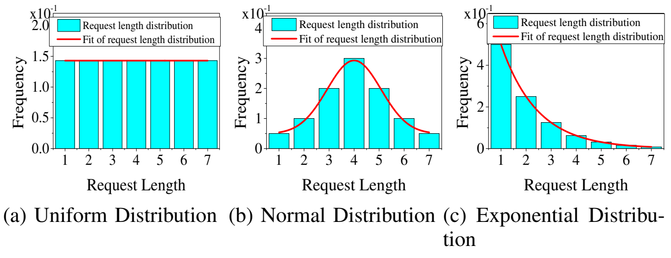
4) *Impact of Mobility and Request Prediction:* As discussed in Section V, we take the prediction of the users' mobility and requests as part of the observation. We conduct the simulations on the two datasets to evaluate the impact of mobility and request prediction on performances.

First of all, we analyze the impact of users' mobility prediction on the results. The performances of total latency on two datasets are shown in Fig. 24 and Fig. 28. Compared with MF-ECS, the total latency of MF-NMP is obviously larger. This is because MF-NMP does not consider the users' mobility during the training of the caching strategy, so the total latency will be larger.

Next, the simulations are conducted where the prediction of users' requests is ignored during the agents' training. As shown in Fig. 24 and Fig. 28, we can find that the performance of MF-NRP is worse than that of MF-ECS and MF-NMP. It it worth noting that the performance of MF-NMP is better than that of MF-NRP. This is because in this paper, we consider the mobility of the users when we predict the users' requests, so the effect of ignoring the users' mobility on the result is less than that of ignoring the request prediction. In other words, in the mobile edge caching scenario, the prediction of users' requests is more important than that of users' mobility.

5) *Performances on Different File Popularity Distributions:* We conduct different simulations on the two datasets under different file popularity distributions, which are shown in Fig. 25 and Fig. 29. It is worth noting that in the simulations, the total latency of RANDOM is larger than that of MF-ECS and the hit ratio of RANDOM is lower than that of MF-ECS, so the larger the D-value is, the better the performance of MF-ECS is.

As shown in Fig. 26 and Fig. 30, compared with RANDOM, the proposed caching strategy MF-ECS can achieve better performances in three different distributions. Compared with normal distribution and exponential distribution, MF-ECS can achieve better performances in the case of uniform distribution. This is because in the case of normal distribution and



(a) Uniform Distribution (b) Normal Distribution (c) Exponential Distribution

Fig. 38: Different distributions of user request length.

exponential distribution, the users will request some files with high probabilities, thus the agents can make better caching decisions and the D-value will decrease. With the increase of storage capacity, D-value first increases and then decreases. The reason for this phenomenon is that MF-ECS can better decide which files should be cached in the storage when the storage capacity is insufficient. After the storage space is increased, the performances of different caching strategies are gradually increased, since the users' requests remain unchanged. Similarly, as shown in Fig. 27 and Fig. 31, in terms of hit ratio, MF-ECS can also achieve better performances than RANDOM, which can make better use of neighbor and local storage capacity.

6) Performances When Changing The Number of Users: The performances when changing the number of users on two datasets are shown in Fig. 32 and Fig. 35. It is not difficult to find that the total latency increases with the increase of the number of users, and the hit ratio and neighbor hit ratio decrease with the number of users. This is reasonable because as the number of users increases, the number of requests received by the edge servers also increases; at the same time, the requests made by different users may be different. Therefore, under the limited storage capacity, the local and neighbor edge servers cannot fully meet the users' requests, and need to obtain files from the cloud server, which greatly increases the total latency and decreases the hit ratio.

7) Performances When Changing The Type of Files: The total type of files is the number of contents, which means how many files the users can request. In other words, the edge server needs to decide which files to cache into its own storage according to the type of files. As shown in Fig. 33 and Fig. 36, the total latency increases with the increase of the file types, this is because the increase of file types increases the action space of the agents. However, the limited storage capacity makes the users' requests cannot always be met locally or in the neighborhood, and need to be obtained from the cloud server, which improves the total latency. For the same reason, the total hit ratio and neighbor hit ratio will decrease with the increase of file types, which can be seen from Fig. 33 and Fig. 36.

8) Performances When Changing The Number of Neighbor Edge Servers: The performances of different caching strategies on two datasets when changing the number of neighbor edge servers are shown in Fig. 34 and Fig. 37. In terms of total latency, the total latency decreases gradually with the increase of the number of neighbor edge servers. This is because with the increase of the number of neighbor edge servers, users' requests can be met more locally or in neighbor

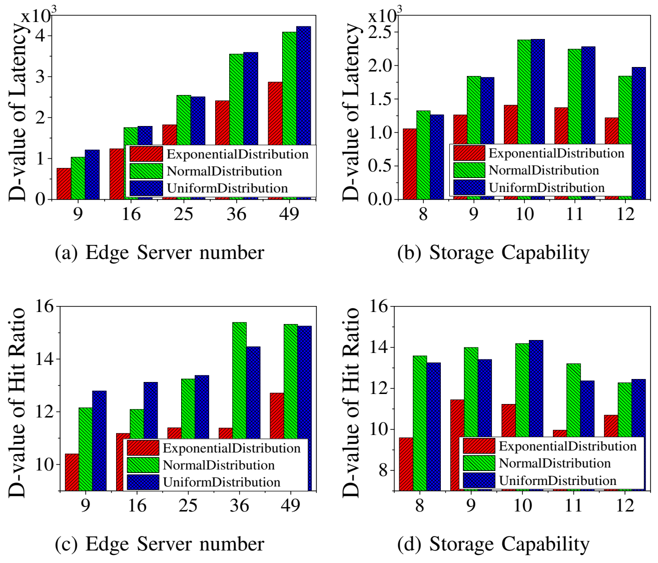


Fig. 39: Performance when changing request length distribution on *roma/taxi*.

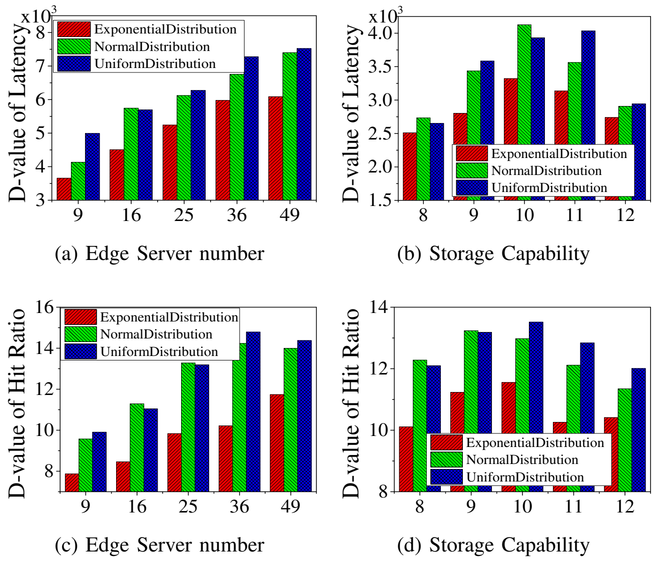


Fig. 40: Performance when changing request length distribution on *EUA*.

edge servers, thus reducing the number of requests for files from the cloud server, then the total latency will be reduced. Accordingly, the total hit ratio and neighbor hit ratio have also increased. From the simulation results, we can see that even if the number of neighbor edge server is not large, the proposed caching strategy MF-ECS still performs better than other caching strategies, and it is appliable.

9) Performances on Different Request Length Distributions: We also conduct simulations under different user request lengths distributions, which are shown in Fig. 38. The user request length indicates the maximum number of files that the user can request in a time slot. As shown in Fig. 39 and Fig. 40, the performance of the proposed caching strategy MF-ECS has little difference under uniform distribution and normal distribution, and the performance difference between MF-ECS and RANDOM strategy is the smallest under exponential distribution. This is because in the case of exponential distribution, most users will only request one file in a time slot, so the number of requests received by each edge server is less than that in other distributions. In this case, the difference between MF-ECS and RANDOM is the smallest. Under the uniform distribution and normal distribution, the edge servers will receive a large number of different requests, so the difference between MF-ECS and RANDOM is large.

VII. Conclusion

In this paper, we investigate the mobile edge caching problem under the storage capacity constraint, in order to minimize the latency of users' requests. To deal with the mobile edge caching problem in the single-time-slot scenario, we first estimate the utility of each file and propose a heuristic caching strategy with an approximation ratio of $\frac{1+k_1k_2}{2k_1}$ to decide which files to be cached in the edge servers. In order to address the edge caching problem for the multiple-time-slot scenario, we propose a caching strategy based on multi-agent deep reinforcement learning called *MF-ECS*, where each edge server is regarded as an agent and decides which files should be cached in its storage. To deal with the cold start problem, we use the proposed heuristic caching strategy to optimize the results during the cold start phase. We conduct extensive simulations based on generated data and three real-world datasets: *roma/taxi* [14], *epfl* [15] and *EUA datasets* [16]. The results show that our proposed caching strategies can achieve the minimum latency compared with other strategies.

References

[1] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2018.

[2] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues," in *Proceedings of Mobile Big Data*, 2015, p. 37–42.

[3] Y. Jararweh, A. Doulat, A. Darabseh, M. Alsmirat, M. Al-Ayyoub, and E. Benkhelifa, "Sdmec: Software defined system for mobile edge computing," in *Proceedings of IEEE International Conference on Cloud Engineering*, 2016, pp. 88–93.

[4] S. Zhang, P. He, K. Suto, P. Yang, L. Zhao, and X. Shen, "Cooperative edge caching in user-centric clustered mobile networks," *IEEE Transactions on Mobile Computing*, vol. 17, no. 8, pp. 1791–1805, 2018.

[5] N. Di, Z. Liu, B. Li, and S. Zhao, "Demand forecast and performance prediction in peer-assisted on-demand streaming systems," in *Proceedings of IEEE INFOCOM*, 2011, pp. 421–425.

[6] H. Li, X. Ma, F. Wang, J. Liu, and K. Xu, "On popularity prediction of videos shared in online social networks," in *Proceedings of the ACM International Conference on Information & Knowledge Management*, 2013, p. 169–178.

[7] J. Xu, V. D. S. Mihaela, J. Liu, and H. Li, "Forecasting popularity of videos using social media," *IEEE Journal of Selected Topics in Signal Processing*, vol. 9, no. 2, pp. 330–343, 2015.

[8] H. Jad, K. Nikhil, and D. Suhas, "Content caching and delivery over heterogeneous wireless networks," in *Proceedings of IEEE INFOCOM*, 2015, pp. 756–764.

[9] A. Khreishah and J. Chakareski, "Collaborative caching for multicell-coordinated systems," in *Proceedings of IEEE INFOCOM*, 2015, pp. 257–262.

[10] F. B. Mismar, B. L. Evans, and A. Alkhateeb, "Deep reinforcement learning for 5g networks: Joint beamforming, power control, and interference coordination," *IEEE Transactions on Communications*, vol. 68, no. 3, pp. 1581–1592, 2020.

[11] F. Tang, Y. Zhou, and N. Kato, "Deep reinforcement learning for dynamic uplink/downlink resource allocation in high mobility 5g hetnet," *IEEE Journal on Selected Areas in Communications*, pp. 1–1, 2020.

[12] T. Chu, J. Wang, L. Codecà, and Z. Li, "Multi-agent deep reinforcement learning for large-scale traffic signal control," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 3, pp. 1086–1095, 2020.

[13] K. Lou, Y. Yang, E. Wang, Z. Liu, T. Baker, and A. K. Bashir, "Reinforcement learning based advertising strategy using crowdsensing vehicular data," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–13, 2020.

[14] L. Bracciale, M. Bonola, P. Loreti, G. Bianchi, R. Amici, and A. Rabuffi, "CRAWDAD dataset roma/taxi (v. 2014-07-17)," Downloaded from https://crawdad.org/roma/taxi/20140717, Jul. 2014.

[15] M. Piorkowski, N. Sarafijanovic-Djukic, and M. Grossglauser, "CRAWDAD dataset epfl/mobility (v. 2009-02-24)," Downloaded from https://crawdad.org/epfl/mobility/20090224, Feb. 2009.

[16] P. Lai, Q. He, M. Abdelrazek, F. Chen, J. Hosking, J. Grundy, and Y. Yang, "Optimal edge user allocation in edge computing with variable sized vector bin packing," *Lecture Notes in Computer Science*, vol. 11236 LNCS, pp. 230 – 245, 2018.

[17] Y. Zhang, M. S. Hossain, A. Ghoneim, and M. Guizani, "Cocme: Content-oriented caching on the mobile edge for wireless communications," *IEEE Wireless Communications*, vol. 26, no. 3, pp. 26–31, 2019.

[18] T. Zhang, X. Fang, Y. Liu, and A. Nallanathan, "Content-centric mobile edge caching," *IEEE Access*, vol. 8, pp. 11 722–11 731, 2020.

[19] C. Zhong, M. C. Gursoy, and S. Velipasalar, "Deep reinforcement learning-based edge caching in wireless networks," *IEEE Transactions on Cognitive Communications and Networking*, vol. 6, no. 1, pp. 48–61, 2020.

[20] F. Wang, F. Wang, J. Liu, R. Shea, and L. Sun, "Intelligent video caching at network edge: A multi-agent deep reinforcement learning approach," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 2499–2508.

[21] K. Zhang, J. Cao, H. Liu, S. Maharjan, and Y. Zhang, "Deep reinforcement learning for social-aware edge computing and caching in urban informatics," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 8, pp. 5467–5477, 2020.

[22] H. Wu, J. Li, J. Zhi, Y. Ren, and L. Li, "Edge-oriented collaborative caching in information-centric networking," in *2019 IEEE Symposium on Computers and Communications (ISCC)*, 2019, pp. 1–6.

[23] M. Sarra, B. Samia, S. Khaled, and D. Mehammed, "New caching system under uncertainty for mobile edge computing," in *2019 Fourth International Conference on Fog and Mobile Edge Computing (FMEC)*, 2019, pp. 129–134.

[24] S. Rahman, M. G. R. Alam, and M. M. Rahman, "Deep learning-based predictive caching in the edge of a network," in *2020 International Conference on Information Networking (ICOIN)*, 2020, pp. 797–801.

[25] G. Qiao, S. Leng, S. Maharjan, Y. Zhang, and N. Ansari, "Deep reinforcement learning for cooperative content caching in vehicular edge computing and networks," *IEEE Internet of Things Journal*, vol. 7, no. 1, pp. 247–257, 2020.

[26] T. Zhao, P. Wang, and S. Li, "Traffic signal control with deep reinforcement learning," in *Proceedings of International Conference on Intelligent Computing, Automation and Systems*, 2019, pp. 763–767.

[27] M. K. Sharma, A. Zappone, M. Assaad, M. Debbah, and S. Vassilaras, "Distributed power control for large energy harvesting networks: A multi-agent deep reinforcement learning approach," *IEEE Transactions on Cognitive Communications and Networking*, vol. 5, no. 4, pp. 1140–1154, 2019.

[28] R. Lowe, Y. WU, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Advances in Neural Information Processing Systems 30*, 2017, pp. 6379–6390.

[29] Y. Yang, R. Luo, M. Li, M. Zhou, W. Zhang, and J. Wang, "Mean field multi-agent reinforcement learning," in *Proceedings of International Conference on Machine Learning*, 2018, pp. 8869–8886.

[30] C. H. Liu, X. Ma, X. Gao, and J. Tang, "Distributed energy-efficient multi-uav navigation for long-term communication coverage by deep reinforcement learning," *IEEE Transactions on Mobile Computing*, vol. 19, no. 6, pp. 1274–1285, 2020.

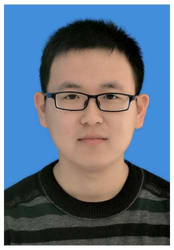
[31] H. Wang, E. Wang, Y. Yang, J. Wu, and F. Dressler, "Privacy-preserving online task assignment in spatial crowdsourcing: A graph-based approach," in *Proc. IEEE INFOCOM*, 2022, pp. 570–579.

[32] H. Wang, Y. Yang, E. Wang, X. Liu, J. Wei, and J. Wu, "Bilateral privacy-preserving worker selection in spatial crowdsourcing," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–14, 2022.

[33] W. Liu, Y. Yang, E. Wang, and J. Wu, "User recruitment for enhancing data inference accuracy in sparse mobile crowdsensing," *IEEE Internet of Things Journal*, vol. 7, no. 3, pp. 1802–1814, 2020.

[34] W. Liu, L. Wang, E. Wang, Y. Yang, D. Zeghlache, and D. Zhang, "Reinforcement learning-based cell selection in sparse mobile crowdsensing," *Computer Networks*, vol. 161, pp. 102–114, 2019.
[35] C. H. Liu, C. Piao, and J. Tang, "Energy-efficient uav crowdsensing with multiple charging stations by deep learning," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 199–208.
[36] D. P. Bertsekas, *Dynamic programming and optimal control*, 1995, vol. 1.
[37] C. H. Liu, T. He, K. Lee, K. K. Leung, and A. Swami, "Dynamic control of data ferries under partial observations," in *Proceedings of IEEE Wireless Communication and Networking Conference*, 2010, pp. 1–6.
[38] M. Lin, W.-J. Hsu, and Z. Q. Lee, "Predictability of individuals' mobility with high-resolution positioning data," in *Proceedings of the ACM Conference on Ubiquitous Computing*, 2012, pp. 381–390.
[39] E. Wang, Y. Yang, J. Wu, W. Liu, and X. Wang, "An efficient prediction-based user recruitment for mobile crowdsensing," *IEEE Transactions on Mobile Computing*, vol. 17, no. 1, pp. 16–28, 2018.
[40] Y. Yang, W. Liu, E. Wang, and J. Wu, "A prediction-based user selection framework for heterogeneous mobile crowdsensing," *IEEE Transactions on Mobile Computing*, vol. 18, no. 11, pp. 2460–2473, 2019.
[41] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org.
[42] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

Yongjian Yang received his B.E. degree in automatization from Jilin University of Technology, Changchun, Jilin, China in 1983; his M.E. degree in computer communication from Beijing University of Post and Telecommunications, Beijing, China in 1991; and his Ph.D. in software and theory of computer from Jilin University, Changchun, Jilin, China in 2005. He is currently a professor and a PhD supervisor at Jilin University, the Vice Dean of the Software College of Jilin University, Director of Key lab under the Ministry of Information Industry, Standing Director of the Communication Academy, and a member of the Computer Science Academy of Jilin Province. His research interests include: network intelligence management, wireless mobile communication and services, and wireless mobile communication.



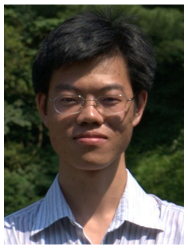
Kaihao Lou received the B.E. degree in software engineering from Jilin University, Changchun, Jilin, China, in 2017, his M.S. degree in computer science and technology from Jilin University, Changchun in 2020. He is currently pursuing the Ph.D. degree in computer science and technology from Jilin University, Changchun, Jilin, China. His current research focuses on mobile crowdsensing and multi-agent reinforcement learning.

En Wang received his B.E. degree in software engineering from Jilin University, Changchun, in 2011, his M.E. degree in computer science and technology from Jilin University, Changchun, in 2013, and his Ph.D. in computer science and technology from Jilin University, Changchun, in 2016. He is currently a Professor in the Department of Computer Science and Technology at Jilin University, Changchun. He is also a visiting scholar in the Department of Computer and Information Sciences at Temple University in Philadelphia. His current research focuses on the efficient utilization of network resources, scheduling and drop strategy in terms of buffer-management, energy-efficient communication between human-carried devices, and mobile crowdsensing.

Wenbin Liu received the B.S. degree in physics and the Ph.D. degree in computer science and technology from Jilin University, China, in 2012 and 2020, where he is currently a Postdoctoral Researcher in Dingxin Scholar Program with the College of Computer Science and Technology. He was also a visiting Ph.D. student in the Wireless Networks and Multimedia Services Department, Telecom SudParis/Institut Mines-Telecom, France. His research interests include Mobile CrowdSensing, Mobile Computing, and Ubiquitous Computing.

Jianwen Shang received the B.E. degree in computer science and technology from Jilin University, Changchun, Jilin, China, in 2020. He is currently pursuing the M.S. degree in computer science and technology from Jilin University, Changchun, Jilin, China. His current research focuses on mobile edge computing.

Xueting Song received the B.E. degree in software engineering from Jilin University, Changchun, Jilin, China, in 2020. She is currently pursuing the M.S. degree in computer science and technology from Jilin University, Changchun, Jilin, China. Her current research focuses on mobile edge computing.



Dawei Li is an Assistant Professor in the Department of Computer Science, Montclair State University. He received the Ph.D. degree in the Department of Computer and Information Sciences, Temple University in 2016. He received the Bachelor's degree from the Advanced Class, Department of Electronics and Information Engineering, Huazhong University of Science and Technology, Wuhan, Hubei, People's Republic of China. His research interest includes energy-aware task scheduling on multi-cores/multiprocessors, data center networks, cloud computing, and big data processing.

Jie Wu is the Director of the Center for Networked Computing and Laura H.Carnell professor at Temple University. He also serves as the Director of International Affairs at College of Science and Technology. He served as Chair of Department of Computer and Information Sciences from the summer of 2009 to the summer of 2016 and Associate Vice Provost for International Affairs from the fall of 2015 to the summer of 2017. Prior to joining Temple University, he was a program director at the National Science Foundation and was a distinguished professor at Florida Atlantic University. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. Dr. Wu regularly publishes in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including IEEE Transactions on Mobile Computing, IEEE Transactions on Service Computing, Journal of Parallel and Distributed Computing, and Journal of Computer Science and Technology. Dr. Wu was general co-chair for IEEE MASS 2006, IEEE IPDPS 2008, IEEE ICDCS 2013, ACM MobiHoc 2014, ICPP 2016, and IEEE CNS 2016, as well as program co-chair for IEEE INFOCOM 2011 and CCF CNCC 2013. He was an IEEE Computer Society Distinguished Visitor, ACM Distinguished Speaker, and chair for the IEEE Technical Committee on Distributed Processing (TCDP). Dr. Wu is a CCF Distinguished Speaker and a Fellow of the IEEE. He is the recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award.