Delay-Aware Service Caching in Edge Cloud: a Adversarial Semi-Bandits Learning-Based Approach

Jinpeng Li, Yunni Xia, Xiaoning Sun, Peng Chen and Xiaobo Li

March 18, 2024

Delay-Aware Service Caching in Edge Cloud: A Adversarial Semi-Bandits Learning-based Approach

1st Jinpeng Li
Collage of Computer Science
Chongqing University
Chongqing, China
705320993@qq.com

2nd Yunni Xia
Collage of Computer Science
Chongqing University
Chongqing, China
xiayunni@hotmail.com

3rd Xiaoning Sun
School of Computer and Information Science
Chongqing Normal University
Chongqing, China
Yunnan Key Laboratory of Service Computing
Yunnan University of Finance and Economics, Kunming
China
sxiaoning@hotmail.com

4th Peng Chen
School of Computer and Software Engineering
Xihua University
Chengdu, China
chenpeng@mail.xhu.edu.cn

5th Xiaobo Li
Chongqing Animal Husbandry Techniques Extension Center
Chongqing, China
202214021065t@stu.cqu.edu.cn

***Abstract*—Mobile Edge Computing (MEC) is an emerging computing paradigm that offloads cloud center functions to the edge server. In a MEC environment, edge servers' limited storage and processing capacity require selective service caching, where only a part of required content can be placed directly upon the destination edge server and the remaining at remote cloud end. A primary challenge in this context is the creation of an effective and responsive service caching algorithm that improves the Quality of Service (QoS) perceived by users while reducing operational costs. This study applies an $M/G/1$ queuing model as the foundational framework and transforms the service caching problem as an adversarial semi-bandit problem. We propose a delay-aware Genetic-Follow-the-Regularized-Leader (GFRL) algorithm, which is capable of guiding decentralized caching decisions. Experimental results indicate that GFRL outperforms traditional methods across various performance metrics.**

***Index Terms*—Mobile edge computing, service caching, adversarial semi-bandits, queuing theory, genetic algorithm**

I. Introduction

As an emerging computational paradigm, MEC is evolving rapidly to meet the challenges posed by the swift advancement of the Internet of Things (IoT), the extensive deployment of 5G communications, and the burgeoning demand for real-time, low-latency processing [1]. The core idea of MEC is to bring computational power to the requestor side, enabling proximity-based provisioning of computation resources in the context of big data applications. Compared to traditional cloud computing, where long delay and low system responsiveness can often be experienced by resource requestors, MEC shifts task processing from the centralized cloud to the edge, such as IoT devices, end-user devices, and edge servers. Such distribution enhances responsiveness, thereby effectively exploiting limited bandwidth provided and guaranteeing low latency [2]. Moreover, 5G base stations are densely deployed [3] with projections of approximately 40-50 stations per square kilometer. Such density offers an ideal infrastructure for the integration and deployment of edge servers.

Fetching the content directly from its producer through traditional connections can usually bring high acquisition delays and error likelihood. In the MEC context, the requested content can be partly placed in the end consumers' proximity for enhanced quality of experience (QoE) to reduce the fetching latency and favor its reuse. Nevertheless, due to capacity constraints, edge-end terminals and servers are often too weak to accommodate all required content. Hence, effective and collaborative cloud-edge caching mechanisms are in high need to manage content placement for multiple mobile users. Moreover, the growing complexity of traffic patterns associated with rapidly time-varying communication channels and the high mobility of the users have made allocating caching resources in a mobile environment extremely challenging due to intermittent communications [4].

In this article, we investigate the service caching challenge in MEC and introduce a delay-aware adversarial semi-bandits approach for dynamically generating service caching schedules. The main contributions are outlined as follows:

1) To minimize user request latency under long-term energy constraints, we model the service caching problem in MEC as an $M/G/1$ queuing model. The model captures long-term energy consumption and provides a fundamental framework for estimating the effectiveness and

This work was supported in part by the Foundation of Yunnan Key Laboratory of Science Computing (No. YNSC23104), in part by the Science Foundation of Chongqing under Grant CSTB2023NSCQ-MSX0782 and Grant KJQN202300533, and in part by the Science and Technology Program of Sichuan Province under Grant No.24NSFTD0025

performance of caching strategies.

2) We develop and implement a decentralized decision-making mechanism, termed GFRL, which utilizes Follow-the-Regularized-Leader(FTRL) and genetic strategies for updates service caching schedules dynamically.
3) We derive the regret upper bound of GFRL.
4) We conduct extensive simulations to validate our proposed method.

II. Related Work

Task offloading is a pivotal topic in MEC research [5] [6] and gains considerable research attention from both academy and industry. There are three types of service caching techniques. The first and most common one is proactive service caching. In proactive service caching, the contents are cached before the request of users. The user's requested contents are predicted using the previous request history, mobility patterns, and learning user preferences. The second is reactive service caching, which caches the content after a user sends a request. There is no such prediction involved in reactive caching. In co-operative service caching, different caching entities cooperate to fulfill the user's demands of service [7].

Recently, machine learning-based methods demonstrated high potency in the area of service caching. Chen *et al.* [8] devised a strategy network using an encoder-decoder model to address computational service placement, employing an on-policy reinforcement method for training. Ke *et al.* [9] proposed a decentralized model-free deep reinforcement learning-based service caching optimization strategy (DDSCOP) to minimize long-term weighted average costs. Huang *et al.* [10] introduced the Independent Learners Service Caching Scheme (ILSCS) utilizing stateless Q-learning for optimal service caching scheme discovery. Wei *et al.* [11] developed a geometric model to predict user mobility and used a Back Propagation (BP) neural network for online prediction of popular services. Hao *et al.* [12] proposed an enhanced deep Q-network-based service placement algorithm for optimal resource allocation through convex optimization. Moreover, the potential of multi-armed bandits algorithms has been exploited. Ou *et al.* [13] described dynamic service placement with limited system information as a contextual multi-armed bandits learning problem, utilizing an online learning algorithm based on Thompson sampling. Han *et al.* [14] combined generalized global bandit with standard multi-armed bandit to address service area overlap issues. Su *et al.* [15] converted the multi-base station caching optimization problem into a resource-constrained multi-agent multi-armed bandit problem, solving it with online learning and cache rounding algorithms. Malazi *et al.* [16] proposed the Distributed Combinatorial Contextual Multi-Armed Bandit (DCC-MAB) method, using UCB as the core algorithm.

Considerable attention is paid on collaborative caching and offloading. Xu *et al.* [17] pioneered jointly optimizing these two aspects, developing an online algorithm based on Lyapunov optimization and Gibbs sampling. Yao *et al.* [18] introduced a graph attention-based multi-agent reinforcement learning (GatMARL) algorithm for optimal strategy learning in edge networks. Wang *et al.* [19] formulated the joint optimization problem as a Markov decision process, proposing a scheme based on the Double Deep Q-Network (DDQN) algorithm. Chen *et al.* [20] tackled the problem as a mixed-integer nonlinear programming issue, applying a Deep Deterministic Policy Gradient (DDPG) algorithm. Furthermore, to address service demand fluctuations and user distribution changes, Wang *et al.* [21] proposed a dynamic server switching algorithm. It targets at reducing the energy cost of network domains.

III. System Models and Problem Formulation

A. System model

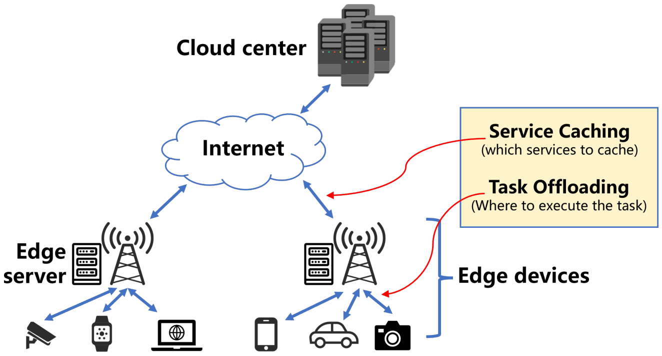


Fig. 1: MEC system model.

As shown in Figure 1, we consider that a MEC environment comprises multiple mobile user devices, multiple edge servers, and a cloud center. The environment comprises n groups of base station and a set of edge servers $B = \{es_1, es_2, \ldots, es_n\}$. A task offloading algorithm assigns tasks from users' devices to computing nodes. The service caching algorithm updates the caching schemes of edge servers according to their storage capacity limitations. We use c_i to denote the storage capacity of the edge server es_i. In environments with high task loads, edge servers are presumed to operate at maximum computing capacity, where f_i indicates the maximum computing power of edge server es_i.

The set of services is $S = \{s_1, s_2, s_3, \ldots, s_l\}$, where the j-th service in this set is s_j and $|S| = l$ is the number of services. Due to the capacity constraint, edge servers offer only a part of services, whereas cloud center provides all l services. For s_j, its corresponding task workload (measured in CPU cycles) is assumed to follow an exponential distribution with a mean of g_j. The notation d_j represents the storage space required. Tasks generated by mobile devices can be processed either locally or transmitted to servers (edge or cloud) for processing. All key symbols used in this paper are summarized in Table I.

TABLE I: Notion table

Variable	Description
n	The number of edge servers
B	A set of edge servers
es_i	The i-th edge server
c_i	The capacity of caching services for es_i
f_i	The maximum power of es_i
l	The number of services
S	A set of services
g_j	The distribution average of the exponential distribution of workload for tasks with service s_j
d_j	The storage space required for service s_j
λ_i^t	The total tasks of users connected to es_i
$\tilde{\lambda}_i^t$	The amount of tasks offloaded to cloud in λ_i^t
$\dot{\lambda}_i^t$	The amount of tasks offloaded to es_i in λ_i^t
$\hat{\lambda}_i^t$	The amount of tasks offloaded to client in λ_i^t
F_i	The task offloading decision of es_i
$a_{i,j}^t$	A binary variable indicates whether service s_j is cached on es_i at time t
$\boldsymbol{a_i^t}$	The caching decision for es_i at time t
b_i^t	The proportion of tasks that can be processed on es_i at time t
$\tilde{\mu}^t$	The total delay required by the cloud center to process each unit task
$\dot{\mu}^t$	The transmission delay for tasks directed to the edge server
$\hat{\mu}^t$	The delay per unit task processed on the user side
e_i	The unit energy consumption of es_i for processing tasks
$E_i^t(\boldsymbol{a_i^t})$	The energy consumption of es_i at time t
$T_i^t(\boldsymbol{a_i^t})$	The average delay of a single task on es_i at time t
$D_i^t(\boldsymbol{a_i^t})$	The response time of a single task on es_i at time t

B. Task Offloading and Service Caching Model

Task Offloading Model: In the coverage area of a base station, users connect to it via a wireless network. We use λ_i^t to represent the total task volume generated by users connected to the base station es_i at time t. Particularly, $\lambda_{i,j}^t$ denotes the task volume with service s_j in λ_i^t. User-generated tasks can either be processed locally or offloaded to servers due to the limited processing capacity of user devices. The task volumes allocated to user devices, edge servers, and cloud servers at time t are denoted by $\hat{\lambda}_i^t$, $\dot{\lambda}_i^t$, and $\tilde{\lambda}_i^t$, respectively. This allocation satisfies the equation:

$$\hat{\lambda}_i^t + \dot{\lambda}_i^t + \tilde{\lambda}_i^t = \lambda_i^t \tag{1}$$

Tasks that exceed the processing capabilities of edge servers are transferred to the cloud. We define $F_i = \{(\hat{\lambda}_i^1, \dot{\lambda}_i^1, \tilde{\lambda}_i^1), (\hat{\lambda}_i^2, \dot{\lambda}_i^2, \tilde{\lambda}_i^2), ...\}$ to represent all task offloading decisions at es_i.

Service Caching Model: Edge servers equip caching capabilities, allowing user-generated tasks to be executed on the edge server. However, due to storage constraints, edge servers can only cache a limited selection of services, thus restricting their ability to process all task types. Consequently, each edge server must make and update judicious caching decisions regularly. The status of service s_j being cached on es_i at time t is denoted by $a_{i,j}^t$, where $a_{i,j}^t = 1$ signifies that the service is cached, and otherwise, it is not. The service caching decision of es_i can be represented as the vector $\boldsymbol{a_i^t} = a_{i,1}^t, a_{i,2}^t, ..., a_{i,l}^t$. We use $b_i^t = \frac{\sum_{j=1}^{l} a_{i,j}^t \dot{\lambda} i,j^t}{\sum_{j=1}^{l} \dot{\lambda} i,j^t}$ to represent the ratio of tasks at time t that can be processed on the edge server, with remaining tasks offloaded to the cloud center. We posit that the caching system operates on containerization technologies, like Docker. Consequently, the caching decisions comply with the constraint $\sum_{j=1}^{l} a_{i,j}^t d_j \leq c_i$.

C. Adversarial Semi-bandit Problem Setting

The service caching problem can be conceptualized as an adversarial semi-bandit problem. In the adversarial environment, the edge server acts as the learner, and each of service corresponds to a fixed arm. We define composite actions as combinations of different arms and establish a predetermined set of these actions for the learner, denoted as $\Theta \subset \{0,1\}^l$. At each moment $t = 1, 2, ...$, the learner selects an action $A_t \in \Theta$ (for es_i, $A_t = \boldsymbol{a_i^t}$), and the environment generates a loss vector $\ell_t \in [-1, 1]^l$. In a semi-bandit environment, the learner only observes the loss associated with each arm in the chosen subset. Specifically, at time t, the incurred loss is $< A_t, \ell_t >$, with the learner receiving feedback $o_t = A_t \circ \ell_t$. Given resource limitations, edge servers are restricted to selecting an action set of a predetermined size, dependent on a specific parameter.

In the adversarial bandits environment, we hypothesize the presence of an adversary who monitors the player's choices. This adversary's actions are unpredictable and potentially hostile, capable of imposing arbitrary losses, possibly influenced by the learner's previous actions and their internal randomness. In service caching, this translates to environmental and user behavior uncertainties, such as variations in user request patterns, network environment fluctuations, or malicious attacks. In other words, the l arms lack a stable reward distribution. This unpredictability means the learner cannot solely depend on historical rewards to forecast future rewards.

The pseudo-regret is the gap between the learner‘s selection and the optimal solution:

$$\overline{\text{Regret}}_{t'} := \mathbb{E}\left[\sum_{t=1}^{t'} \langle A_t - a^*, \ell_t \rangle\right] \tag{2}$$

where $a^* = \arg\min_{a \in \Theta} \mathbb{E}\left[\sum_{t=1}^{t'} \langle a, \ell_t \rangle\right]$ represents the optimal combinatorial action, and the expectation is based on the stochastic behaviors of both the learner's actions and the environmental responses.

D. Energy Cost and Task Response Delay

In an environment where the task load and server computing capabilities are unchangeable, we aim to minimize energy consumption and task response time by optimizing task offloading and service caching algorithms.

Energy Cost: Energy consumption represents a significant cost for operators. Edge servers incur a baseline energy consumption, called standby energy, even without computational tasks, denoted as v_i^t for es_i. The energy required during computational tasks depends on the task load and per-unit energy consumption. As noted earlier, edge servers function at maximum power while processing tasks. Consequently, the total energy consumption for es_i can be articulated as follows:

$$E_i^t(\boldsymbol{a_i^t}) = v_i^t + e_i \sum_{j=1}^{l} g_j b_i^t \dot{\lambda}_{i,j}^t \tag{3}$$

where e_i represents the unit energy consumption of es_i while operating at its maximum power f_i, and $\sum_{j=1}^{l} g_j b_i^t \dot{\lambda}_{i,j}^t$ the cumulative number of CPU cycles needed for es_i to process the tasks.

Task Response Delay: We use $\hat{\mu}$ to denote the delay per unit task processed on the user side. Since the task does not need to be transmitted, $\hat{\mu}$ is primarily determined by the computation time. $\tilde{\mu}$ represents the total delay required by the cloud center to process each unit task, including task transmission and computation time. The transmission delay for tasks directed to the edge server, denoted by $\dot{\mu}$, is computable using the Shannon formula. For service time calculation post-task arrival at the edge server, we adopt the $M/G/1$ queue model. Let r denote the random variable for service time, with its expected value given by:

$$\mathbb{E}[r] = \sum_{j=1}^{l} g_j \dot{\lambda}_{i,j}^t / f_i \dot{\lambda}_i^t \tag{4}$$

where $\dot{\lambda}_i^t = \sum_{j=1}^{l} \dot{\lambda}_{i,j}^t$, $\dot{\lambda}_{i,j}^t / \dot{\lambda}_i^t$ represents the proportion of tasks of service s_j in the total task volume, and $\mathbb{E}[r^2] = \sum_{j=1}^{l} g_j^2 \dot{\lambda}_{i,j}^t / f_i^2 \dot{\lambda}_i^t$. According to the Pollaczek-Khinin formula [22], the expected residence time of a single task is expressed as follows:

$$\begin{aligned} T_i^t(\boldsymbol{a_i^t}) &= \mathbb{E}[r] + \frac{\dot{\lambda}_i^t b_i^t \mathbb{E}[r^2]}{2 - 2\dot{\lambda}_i^t b_i^t \mathbb{E}[r]} \\ &= \mathbb{E}[r] + \frac{\sum_{j=1}^{l} a_{i,j}^t \dot{\lambda}_{i,j}^t \mathbb{E}[r^2]}{2 - 2\dot{\lambda}_i^t b_i^t \mathbb{E}[r]} \end{aligned} \tag{5}$$

The task response delay on the edge server comprises three components: computation delay, transmission delay, and additional delay due to a cache miss. The total delay is expressed as:

$$D_i^t(\boldsymbol{a_i^t}) = \dot{\lambda}_i^t b_i^t T_i^t(a^t) + \dot{\lambda}_i^t \dot{\mu} + \dot{\lambda}_i^t (1 - b_i^t) \tilde{\mu}^t \tag{6}$$

E. Problem Formulation

In the MEC environment, network operators aim to minimize user request latency while maintaining the energy consumption of edge servers at an acceptable level. The objective can be formulated as follows:

$$(\mathbf{P1}) \quad \min_{a^t, \forall t} \quad \lim_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} \sum_{i=1}^{n} \left[D_i^t(\boldsymbol{a_i^t}) + \hat{\lambda}_i^t \hat{\mu}^t + \tilde{\lambda}_i^t \tilde{\mu}^t \right] \tag{7}$$

$$\begin{aligned} s.t. \quad & \mathbf{C1.} \quad \lim_{T \to \infty} \frac{1}{T} \sum_{t=1}^{T} \sum_{i=1}^{n} E_i^t(\boldsymbol{a_i^t}) \le Q \\ & \mathbf{C2.} \quad E_i^t(\boldsymbol{a_i^t}) \le E_i^{max} \qquad \forall t, \forall i \\ & \mathbf{C3.} \quad D_i^t(\boldsymbol{a_i^t}) \le D_i^{max} \qquad \forall t, \forall i \\ & \mathbf{C4.} \quad \sum_{j=1}^{l} a_{i,j}^t d_j \le c_i \qquad \forall t, \forall i \\ & \mathbf{C5.} \quad \mathbb{E}[\langle X_t - x^*, \ell_t \rangle] < Reg_t^{max} \qquad \forall t \end{aligned}$$

$\boldsymbol{C1}$ is the long-term energy constraint of edge servers, with Q as the upper limit of long-term energy. $\boldsymbol{C2}$ and $\boldsymbol{C3}$ impose constraints on the energy and task request delay for each time slot, respectively, where E_i^{max} and D_i^{max} are the upper limits for energy and request delay. $\boldsymbol{C4}$ represents the storage capacity limitation of edge servers. $\boldsymbol{C5}$ reflects that the regret at any moment in the adversarial environment should not exceed Reg_t^{max}.

The primary difficulty in deriving the optimal solution for this problem stems from the unpredictability of future data. To determine the optimal solution **P1**, it is necessary yet challenging to forecast the task demand distribution for all future time points. In fact, the **P1** problem resembles the Capacitated Facility Location Problem (CFLP), which is a NP-hard problem. Therefore, we need an online method to leverage available information for real-time decision-making regarding task offloading and service caching.

IV. The Proposed Method

A. GFRL Algorithm

To guarantee the practicality of our algorithm, we discretize time into intervals corresponding to the scale of task offloading and cache updating. At the start of each interval, Algorithm 1 allocates tasks to the user end, edge server, or cloud server and calculates the task response delay using the $M/G/1$ model, which forms the basis for ℓ_t (Line 3). Cache updates occur at predetermined intervals based on the parameter t_{in}. When a caching update is needed, the necessity of revising o_{max} and $A_{max}$is assessed (Lines 5-8), followed by the computation of the regularized leader $a_t = \underset{a \in \mathrm{Conv}(\Theta)}{\arg\min} \langle a, \hat{\mathcal{L}}_{t-1} \rangle + \epsilon_t^{-1} E(a)$, where Conv$(\Theta)$ is the convex hull of Θ, $\hat{\mathcal{L}}_{t-1} = \sum_{s=1}^{t-1} \hat{\ell}_s$ the cumulative estimated loss, ϵ_t a learning rate, $E(a)$ a regularizer that maps elements from Conv(Θ) to $\mathbb{R} \cup \{+\infty\}$ (Line 10). Tsallis and Shannon entropies are employed in constructing E as follows: [23]:

$$E(a) = \sum_{i=1}^{l} -\sqrt{a_i} + \gamma(1 - a_i) \log(1 - a_i) \tag{8}$$

Then the algorithm samples A_t from $Y(a_t)$ which is a distribution over Θ satisfying $\mathbb{E}_{A \sim Y(a)}[A] = a$. An efficient sampling rule Y is always achievable in our setting(refer to Section IV-C for an example). The algorithm employs a modified genetic algorithm to crossbreed A_t and A_{max} for enhancing performance (Line 12). Finally, the MEC server caches the services in A_t and communicates the caching strategy to neighboring servers(Line 13). Task allocating in the subsequent time interval is based on the updated caching strategy.

Additionally, irrespective of caching strategy changes, $\hat{\mathcal{L}}_t$ is updated in each interval (Lines 18-20) by: 1) Calculating the Hadamard product of A_t and ℓ_t to obtain o_t, 2) constructing unbiased loss-estimators $\hat{\ell}_t$ where $\hat{\ell}_{ti} = \frac{(o_{ti}+1)\mathbb{1}_t(i)}{a_{ti}} - 1$ for all i , with $\mathbb{1}_t(i)$ as an indicator function that equals one if $A_{ti} = 1$ and 0 otherwise, 3) accumulating the cumulative estimated loss $\hat{\mathcal{L}}_t$.

Algorithm 1: GFRL algorithm for an edge server

```
   Input: time interval $t_{in}$, task offloading strategy $F_n$,
          $0<\gamma \leq 1$, sampling scheme $Y$
   Output: caching decision $A_t$
 1 Initialize $\hat{\mathcal{L}}_0 = (0, ..., 0)$, $\epsilon_t = 1/\sqrt{t}$, $o_{max} = 0$,
     $A_{max} = 0$
 2 foreach episode do
 3     task offloading according to $F_n$
 4     if $t \bmod t_{in} == 0$ then
 5         if $o > o_{max}$ then
 6             $A_{max} \leftarrow A_t$
 7             $o_{max} \leftarrow o$
 8         end
 9         initialize $o \leftarrow 0$
10         compute $a_t = \underset{a \in \text{Conv}(\Theta)}{\arg\min} \langle a, \hat{\mathcal{L}}_{t-1} \rangle + \epsilon_t^{-1} E(a)$
11         sample $A_t \sim Y(a_t)$
12         genetic_crossover()
13         cache $A_t$ into the server and synchronize
             information from adjacent servers
14     else
15         $a_t \leftarrow a_{t-1}$
16         $A_t \leftarrow A_{t-1}$
17     end
18     observe $o_t = A_t \circ \ell_t$
19     construct estimator $\hat{\ell}_t, \forall i : \hat{\ell}_{ti} = \frac{(o_{ti}+1)\mathbb{1}_t(i)}{a_{ti}} - 1$
20     update $\hat{\mathcal{L}}_t = \hat{\mathcal{L}}_{t-1} + \hat{\ell}_t$
21     $o \leftarrow o + o_t$
22 end
```

B. Modified Genetic Algorithm

To improve the performance of our algorithm, we incorporate a modified genetic algorithm. The algorithm inputs the current caching decision A_t and the current optimal decision A_{max}, subsequently generating a revised A_t. The detailed steps, as outlined in Algorithm 2, include: 1)Identifying services present in A_t but absent in A_{max} (Lines 2-6); 2) Identifying services in A_{max} but missing in A_t (Lines 7-11), and 3) Allocating them into two sets, $S1$ and $S2$, respectively. Each service in $S2$ is then evaluated for its potential to replace a service in A_t, governed by the parameter a.

Algorithm 2: Modified Genetic Algorithm

```
   Input: services list $A_t$, pre-optimal strategy $A_{max}$,
          probability parameter $a$
   Output: caching decision $A_t$
 1 Initialize $S_1 \leftarrow \emptyset$, $S_2 \leftarrow \emptyset$
 2 foreach $q \in A_t$ do
 3     if $q \notin A_{max}$ then
 4         Put $q$ into $S_1$
 5     end
 6 end
 7 foreach $q \in A_{max}$ do
 8     if $q \notin A_t$ then
 9         Put $q$ into $S_2$
10     end
11 end
12 foreach $s \in S_2$ do
13     $r \leftarrow Random()$
14     if $r < a$ then
15         Remove $S_1[0]$ form $A_t$
16         Remove $S_1[0]$ form $S_1$
17         $A_t \leftarrow A_t \cup \{s\}$
18     end
19 end
20 return $A_t$
```

C. Example of Rule Y and Regret

In our environment, the learners are constrained to select a fixed-size action set. Let $m \in \{1, ..., l\}$ be the fixed parameter, and the action set is defined as:

$$\Theta = \{a \in [0, 1]^l \mid \sum_{i=1}^{l} a_i = m\} \tag{9}$$

Prior to given the sampling rule Y, we define auxiliary vectors $\alpha_{i,j}$ and a uniform sampling rule $Y_{i,j}$ for $i \in [0, m]$ and $j \in [0, l - m]$:

$$\alpha_{i,j} = \left(\underbrace{1, \dots, 1}_{i}, \frac{m-i}{l-i-j}, \dots, \frac{m-i}{l-i-j}, \underbrace{0, \dots, 0}_{j}\right) \in \text{Conv}(\Theta) \tag{10}$$

$$Y_{i,j} = \text{Uniform}\left(\{a \in \Theta \mid a_{1,\dots i} = 1 \wedge a_{l-j+1,\dots l}\} = 0\}\right) \tag{11}$$

The convex hull of Θ can always be described by a polynomially bounded set of constraints. Therefore, for a combinatorial action $a' \in \text{Conv}(\Theta)$, the sampling rule Y that satisfies $\mathbb{E}_{A \sim Y_{(a')}}[A] = a'$ can be obtained in the following steps:

1) *Sorting*: We start by sorting the components of a' in descending order to get a new vector a, where $a_1 \geq$

$a_2 \geq \ldots \geq a_l$.The ordering is conducted along the coordinate dimension of the vector (uni-dimensional sorting), thereby allowing completion within $O(l\log(l))$ time.

2) *Decomposition*: Next, we decompose a into $\sum_{s=0}^{l} y_{a,s}\alpha_{i_s,j_s}$, where $y_{a,s} \in [0,1]$ $\forall s \in \{0,\ldots,l\}$ and $\sum_{s=0}^{l} y_{a,s} = 1$. For α_{i_s,j_s}, with indexing starting from $(0,0)$, as s transitions to $s+1$, either i or j increments by one. From a mathematical perspective, $(i_0, j_0) = (0,0)$ and $(i_{s+1}, s_{s+1}) - (i_s, j_s) \in \{(1,0),(0,1)\}$. This decomposition can be greedily computed in $O(l)$ time.
3) *Sampling Rule*: The complete sampling rule is a combination of above steps, formally written as $\sum_{s=0}^{l} y_{a,s}Y_{i_s,j_s}$, where $y_{a,s}$ and (i_s, j_s) originate from step IV-C.

Step 1) dominates the runtime, resulting in an overall time complexity of O(l log(l)). Moreover, in the context of $Eq.(9)$, the pseudo-regret of GFRL with

$$\gamma = \begin{cases} 1 & \text{if } m \leq \frac{l}{2} \\ \min\left\{1, \frac{1}{\sqrt{\log(l/(l-m))}}\right\} & \text{otherwise,} \end{cases} \qquad (12)$$

meets

$$\overline{\text{Regret}}_{t'} \leq \begin{cases} O\left(\sqrt{mlt'}\right) & \text{if } m \leq \frac{l}{2} \\ O\left((l-m)\sqrt{\log\left(\frac{l}{l-m}\right)t'}\right) & \text{otherwise} \end{cases} \qquad (13)$$

Zimmert $et\ al$. [23] have made efforts on the pseudo-regret and the optimality.

V. Performance Evaluation

A. Experiment Setting

To simulate a real-world environment, we integrate the Shanghai Telecom base station dataset with a comprehensive dataset [24]. The merged dataset contains 7.2 million request records from 3233 edge nodes and 9481 users. It enables us to track user network access, including request timings and designated base stations.To challenge the algorithm's decision-making, we introduced random attackers with a 1% appearance probability, although they do not interfere with the loss in every round. The round-trip time to the cloud server is set at 74 milliseconds. The cache updating parameter t_{in} is fixed at 5, indicating service caching updates every five epochs. We use an iterative method to determine the value of a_t and apply the Hessian matrix to accelerate this iterative process.

The GFRL algorithm was implemented in Python 3.10 and tested on a computer equipped with AMD Ryzen 7 6800H 3.20 GHz processor and 16.0 GB RAM.

B. Comparison Algorithms

To evaluate the performance of the GFRL algorithm, we selected the following four benchmarks:

1) *Oracle Algorithm:* This algorithm hypothetically has access to all future information, enabling the caching strategy to align with future requests perfectly. It represents an ideal best-case scenario.
2) *DQN-DSP Algorithm [12]:* In this algorithm, the service caching optimization challenge is bifurcated into two components: resource allocation and service placement. The former utilizes convex optimization to derive the optimal solution, while the latter employs Deep Q-network (DQN) to understand and address service requests.
3) *AUSP Algorithm [13]:* This approach conceptualizes service caching as a contextual multi-armed bandit problem. Caching decisions are made using a lightweight online learning algorithm, considering multi-server collaboration scenarios.
4) *DCC-MAB Algorithm [16]:* In this algorithm, each MEC independently operates a MAB and makes autonomous service caching decisions. The algorithm leverages neighboring information as context and utilizes the Upper Confidence Bound (UCB) algorithm for real-time service caching updates.

C. Performance Evaluation

To mitigate the influence of experimental errors on the results, we averaged the data from a minimum of 10 trials to assess the algorithm's performance. Specifically for the AUSP and DCC-MAB algorithms, which exhibited higher result variability, we amplified the number of runs to 100.

Regret Analysis: We initially examined the average pseudo-regret across various environments. As shown in Figure 2, the GFRL's upper bounds align with Equation (13). Here, GFRL-1 corresponds to parameters $l = 100, m = 50$; GFRL-2 to $l = 200, m = 100$; GFRL-3 to $l = 300, m = 150$; and GFRL-4 to $l = 400, m = 200$.

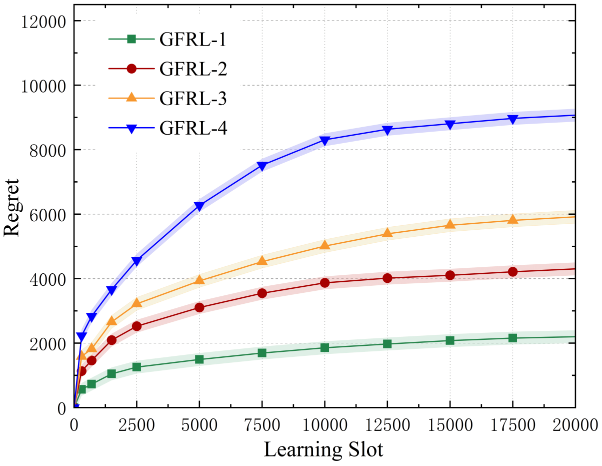
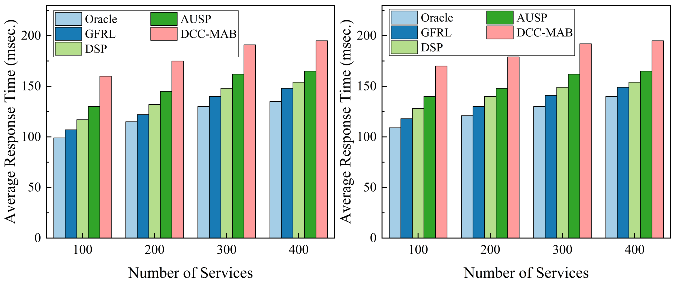


Fig. 2: Total regret under different edge scenarios.

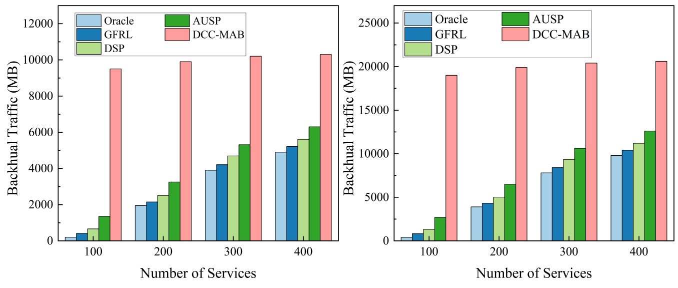
Number of Services: Figure 3 shows that with $T = 20K$ (representing 20,000 tasks), an increase in the number of services generally leads to a rise in the average response time of the algorithms. However, when task volume grows from $20k$



(a)

(b)

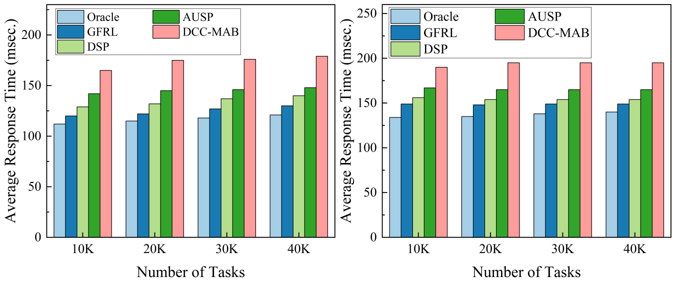
Fig. 3: Number of Services and Average Response Time: (a)T=20k; (b)T=40k.



(a)

(b)

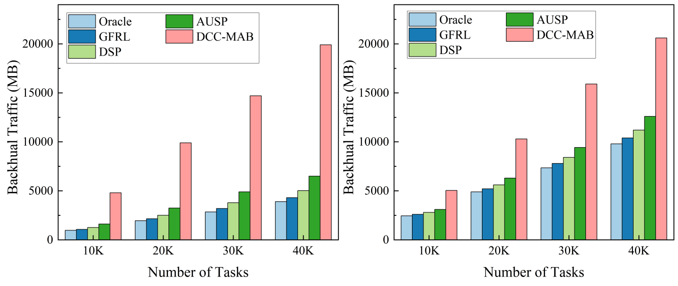
Fig. 4: Number of Services and Backhaul Traffic: (a)T=20k; (b)T=40k.



(a)

(b)

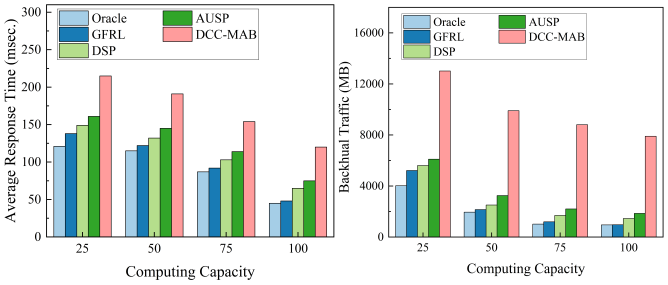
Fig. 5: Number of Tasks and Average Response Time: (a)l=200; (b)l=400.



(a)

(b)

Fig. 6: Number of Tasks and Backhaul Traffic: (a)l=200; (b)l=400.



(a)

(b)

Fig. 7: Impact of Computing Capacity:(a)Computing Capacity and Average Response Time; (b)Computing Capacity and Backhaul Traffic.

to $40k$, the response time of each algorithm increases only marginally, suggesting stable algorithm performance under adequate resources. In scenarios with 100 services, the Oracle method is 7% faster than GFRL, while GFRL outpaces DSP and AUSP by 8% and 10%, respectively. DCC-MAB lags due to its limited adaptability in adversarial environments.

Figure 4 illustrates the average backhaul traffic in different environments. With 400 services, Oracle's traffic is 6% lower than GFRL's, which in turn is 7% and 17% lower than DSP's and AUSP's, respectively. DCC-MAB's performance is notably the least efficient.

Number of Tasks: Figure 5 reveals that the algorithms' average response times remain relatively constant as task numbers increase, given a stable service scale. This stability indicates efficient edge resource utilization by the algorithms, mitigating congestion risks in densely populated environments. Notably, with T=$10K$, GFRL surpasses all other algorithms, except Oracle, by 10-20%.

Figure 6 shows the backhaul traffic trends as task numbers rise. An almost linear increase in backhaul traffic with task scale expansion reflects the algorithms' strategic shifts: prioritizing average response time reduction in low-activity environments and balancing increased backhaul costs in high-activity environments.

Capacity of Edge Servers: Figure 7 shows that as computing capacity increases, both the average response time and backhaul traffic of the algorithms decrease. GFRL's performance is 12% lower than the baseline Oracle but remains the most efficient among the compared algorithms.

VI. Conclusion

In this work, we propose an adversarial method for delay-aware service caching in edge cloud. This method synthesizes FTRL and genetic strategies for developing the GFRL algorithm, which is proved to closely converge to the optimal upper bound. Numerical results demonstrate that the proposed method outperforms traditional algorithms on multiple metrics and is capable of adapting itself to increases of task load with high level of availability. In further research, we intend to: 1) Investigate the joint algorithm of task offloading and service caching. 2) Employ more complex bandits mechanisms to enhance algorithm accuracy. 3) Explore the potential of the muti-layer edge cloud computing architecture.

References

[1] C. Wu, Q. Peng, Y. Xia, Y. Jin, and Z. Hu, "Towards cost-effective and robust AI microservice deployment in edge computing environments," *Future Gener. Comput. Syst.*, vol. 141, pp. 129–142, 2023. [Online]. Available: https://doi.org/10.1016/j.future.2022.10.015

[2] Y. Li, X. Sun, Y. Xia, P. Chen, Y. Li, and Q. Peng, "M-MNFT: A novel modified (m, n)-fault tolerance approach for service migration in vehicular edge computing," in *IEEE International Conference on Software Services Engineering, SSE 2023, Chicago, IL, USA, July 2-8, 2023*, C. A. Ardagna, N. L. Atukorala, C. K. Chang, J. Fan, G. C. Fox, S. Helal, Z. Jin, Q. Lu, T. Seceleanu, and S. S. Yau, Eds. IEEE, 2023, pp. 170–179. [Online]. Available: https://doi.org/10.1109/SSE60056.2023.00031

[3] M. Shafi, A. F. Molisch, P. J. Smith, T. Haustein, P. Zhu, P. De Silva, F. Tufvesson, A. Benjebbour, and G. Wunder, "5g: A tutorial overview of standards, trials, challenges, deployment, and practice," *IEEE journal on selected areas in communications*, vol. 35, no. 6, pp. 1201–1221, 2017.

[4] J. Bi, H. Yuan, S. Duanmu, M. Zhou, and A. Abusorrah, "Energy-optimized partial computation offloading in mobile-edge computing with genetic simulated-annealing-based particle swarm optimization," *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3774–3785, 2020.

[5] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE communications surveys & tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[6] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[7] X. Xiao, Y. Ma, Y. Xia, M. Zhou, X. Luo, X. Wang, X. Fu, W. Wei, and N. Jiang, "Novel workload-aware approach to mobile user reallocation in crowded mobile edge computing environment," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 7, pp. 8846–8856, 2022.

[8] Y. Chen, Y. Sun, B. Yang, and T. Taleb, "Joint caching and computing service placement for edge-enabled iot based on deep reinforcement learning," *IEEE Internet of Things Journal*, vol. 9, no. 19, pp. 19 501–19 514, 2022.

[9] H. Ke, H. Wang, K. Yang, and H. Sun, "Service caching decision-making policy for mobile edge computing using deep reinforcement learning," *IET Communications*, vol. 17, no. 3, pp. 362–376, 2023.

[10] B. Huang, Z. Ran, D. Yu, Y. Xiang, X. Shi, Z. Li, and Z. Xu, "Stateless q-learning algorithm for service caching in resource constrained edge environment," *Journal of Cloud Computing*, vol. 12, no. 1, p. 132, 2023.

[11] H. Wei, H. Luo, and Y. Sun, "Mobility-aware service caching in mobile edge computing for internet of things," *Sensors*, vol. 20, no. 3, p. 610, 2020.

[12] Y. Hao, M. Chen, H. Gharavi, Y. Zhang, and K. Hwang, "Deep reinforcement learning for edge service placement in softwarized industrial cyber-physical system," *IEEE Trans. Ind. Informatics*, vol. 17, no. 8, pp. 5552–5561, 2021. [Online]. Available: https://doi.org/10.1109/TII.2020.3041713

[13] T. Ouyang, X. Chen, Z. Zhou, R. Li, and X. Tang, "Adaptive user-managed service placement for mobile edge computing via contextual multi-armed bandit learning," *IEEE Trans. Mob. Comput.*, vol. 22, no. 3, pp. 1313–1326, 2023. [Online]. Available: https://doi.org/10.1109/TMC.2021.3106746

[14] Y. Han, L. Ai, R. Wang, J. Wu, D. Liu, and H. Ren, "Cache placement optimization in mobile edge computing networks with unaware environment - an extended multi-armed bandit approach," *IEEE Trans. Wirel. Commun.*, vol. 20, no. 12, pp. 8119–8133, 2021. [Online]. Available: https://doi.org/10.1109/TWC.2021.3090440

[15] L. Su, R. Zhou, N. Wang, J. Chen, and Z. Li, "Multi-agent multi-armed bandit learning for content caching in edge networks," in *IEEE International Conference on Web Services, ICWS 2022, Barcelona, Spain, July 10-16, 2022*, C. A. Ardagna, N. L. Atukorala, B. Benatallah, A. Bouguettaya, F. Casati, C. K. Chang, R. N. Chang, E. Damiani, C. G. Guegan, R. Ward, F. Xhafa, X. Xu, and J. Zhang, Eds. IEEE, 2022, pp. 11–16. [Online]. Available: https://doi.org/10.1109/ICWS55610.2022.00018

[16] H. T. Malazi and S. Clarke, "Distributed service placement and workload orchestration in a multi-access edge computing environment," in *IEEE International Conference on Services Computing, SCC 2021, Chicago, IL, USA, September 5-10, 2021*, B. Carminati, C. K. Chang, E. Daminai, S. Deng, W. Tan, Z. Wang, R. Ward, and J. Zhang, Eds. IEEE, 2021, pp. 241–251. [Online]. Available: https://doi.org/10.1109/SCC53864.2021.00037

[17] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 207–215.

[18] Z. Yao, S. Xia, Y. Li, and G. Wu, "Cooperative task offloading and service caching for digital twin edge networks: A graph attention multi-agent reinforcement learning approach," *IEEE Journal on Selected Areas in Communications*, 2023.

[19] L. Wang and G. Zhang, "Joint service caching, resource allocation and computation offloading in three-tier cooperative mobile edge computing system," *IEEE Transactions on Network Science and Engineering*, 2023.

[20] L. Chen, G. Gong, K. Jiang, H. Zhou, and R. Chen, "Ddpg-based computation offloading and service caching in mobile edge computing," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2022, pp. 1–6.

[21] Q. Wang, Q. Xie, N. Yu, H. Huang, and X. Jia, "Dynamic server switching for energy efficient mobile edge networks," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–6.

[22] L. Kleinroch, "Queueing system," *Volume II, J.*, 1976.

[23] J. Zimmert, H. Luo, and C. Wei, "Beating stochastic and adversarial semi-bandits optimally and simultaneously," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 2019, pp. 7683–7692. [Online]. Available: http://proceedings.mlr.press/v97/zimmert19a.html

[24] K. Moghaddasi and M. Masdari, "Blockchain-driven optimization of iot in mobile edge computing environment with deep reinforcement learning and multi-criteria decision-making techniques," *Cluster Computing*, pp. 1–29, 2023.