Computer Networks 250 (2024) 110575

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Learning-driven service caching in MEC networks with bursty data traffic and uncertain delays

Wenhao Ren [a], Zichuan Xu [a,*], Weifa Liang [b], Haipeng Dai [c], Omer F. Rana [d], Pan Zhou [e], Qiufen Xia [a], Haozhe Ren [a], Mingchu Li [a], Guowei Wu [a]

[a] School of Software, Dalian University of Technology, Dalian, 116024, China
[b] Department of Computer Science at City University of Hong Kong, Hong Kong, China
[c] State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, Jiangsu 210023, China
[d] Cardiff University, United Kingdom
[e] School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan, Hubei 430074, China

ARTICLE INFO

Keywords:
Mobile edge computing
Service caching
Bursty data traffic
Uncertain processing delays
Online learning

ABSTRACT

Mobile edge computing (MEC) provides extremely low-latency services for mobile users, by attaching computing resources to 5G base stations in an MEC network. Network service providers can cache their services from remote data centers to base stations to serve mobile users within their proximity, thereby reducing service latencies. However, mobile users of network services usually have bursty requests that require immediate processing in the MEC network. The data traffic of such requests is bursty, since mobile users have various hidden features including locations, group tags, and mobility patterns. Furthermore, such bursty data traffic causes uncertain congestion at base stations and thus leads to uncertain processing delays. Considering the limited resources of base stations, network services may not be able to be placed into base stations permanently to handle the bursty data traffic. As such, network services need to be dynamically cached in the MEC network to fully address the bursty data traffic and uncertain processing delays.

In this paper, we investigate the problem of dynamic service caching and task offloading in an MEC network, by adopting the online learning technique to harness the challenges brought by bursty data traffic and uncertain processing delays. We first propose an online learning algorithm for the problem with uncertain processing delays, by utilizing the multi-armed bandits technique, and analyze the regret bound of the proposed algorithm. We then propose another online learning algorithm for the problem with bursty data traffic and uncertain processing delays, which adaptively learns the bursty data traffic of requests, based on small samples of mobile users' hidden features. We also propose a novel architecture of Generative Adversarial Network (GAN) to accurately predict user demands using small samples of mobile users' hidden features. Based on the proposed GAN model, we then devise an efficient heuristic for the problem with the uncertainties of both bursty data traffic and uncertain delays. We finally evaluate the performance of the proposed algorithms by simulations, using a real data trace. Experimental results show that the performance of the proposed algorithms outperforms existing ones by up to 44% in terms of average delay.

1. Introduction

With the fast development of AI and communication technologies, various mobile users are demanding diverse AI services, such as autonomous driving, smart surveillance, and faulty product identification of Industry 4.0. Mobile Edge Computing (MEC) is envisioned as a key to enable pervasive AI services with guaranteed performance. In an MEC network, we can replicate network services that are originally deployed in remote data centers to base stations (BSs), allowing user requests to be processed at nearby locations and thereby reducing service delays. Due to the limited resource capacities of BSs, replicated service instances can only stay in the MEC network temporarily. To guarantee seamless task processing, the state data generated by each replicated service instance in the MEC network needs to be updated to its original instance in the remote data center. We refer such temporary

* Corresponding author.
E-mail addresses: wenhaoren@mail.dlut.edu.cn (W. Ren), z.xu@dlut.edu.cn (Z. Xu), weifa.liang@cityu.edu.hk (W. Liang), haipengdai@nju.edu.cn (H. Dai), RanaOF@cardiff.ac.uk (O.F. Rana), panzhou@hust.edu.cn (P. Zhou), qiufenxia@dlut.edu.cn (Q. Xia), renhaozhe@mail.dlut.edu.cn (H. Ren), mingchul@dlut.edu.cn (M. Li), wgwdut@dlut.edu.cn (G. Wu).

https://doi.org/10.1016/j.comnet.2024.110575
Received 6 November 2023; Received in revised form 22 March 2024; Accepted 4 June 2024
Available online 7 June 2024
1389-1286/© 2024 Published by Elsevier B.V.

service replication with state data updating from remote data centers to BSs in an MEC network as *service caching* [1,2]. In this paper, we consider the joint service caching and task offloading in an MEC network.

Due to the hidden features of mobile users, such as location, mobility, and group tags, the services serving user requests in an MEC network have bursty data traffic and uncertain processing delays. Specifically, the data traffic of requests in an MEC network is usually unexpected, which is referred to as *bursty data traffic*. For example, a sudden event can rapidly increase user demands in the MEC network [3,4], leading to bursty data traffic for bandwidth resources to data transfer in the network. Also, as shown in [5], the multimedia big data from real-world applications shows that both computing and bandwidth resources are needed to handle bursty data traffic. In other words, it is difficult to grasp the patterns of such bursty data traffic in short time, especially when mobile users simultaneously generate large volumes of data [6]. Due to the bursty data traffic of users, the processing delays of BSs in an MEC network are usually uncertain. The delay incurred in each link (either wireless or wired) of the MEC network can vary, depending on the workload on the link [7].

Addressing the afore-mentioned bursty data traffic and uncertain processing delays in service caching and task offloading in MEC networks poses significant challenges. First, how to jointly cache services for mobile users with bursty data traffic from remote data centers to BSs (or cloudlets) and offload user tasks to cached service instances? Particularly, decoupling service caching from task offloading may result in user tasks with bursty data traffic being offloaded to services with very limited reserved resources, thereby degrading the service performance substantially. What is worse, in the MEC network, there are BSs in the cellular network, switches in the edge backhaul, and data centers in the core network. The complex interplay between BSs, switches and data centers further complicates the service caching and task offloading. Specifically, caching services into the closest BS may guarantee the high responsiveness of user requests. However, the resources in each BS are capacitated, and when they are congested with bursty data traffic, an alternative is to cache services in nearby BSs via backhaul paths/links. It is challenging to select BSs under the uncertain processing delays to alleviate the bursty data traffic of congested BSs such that the average delay experienced by each request is minimized. Second, how to perform performance guaranteed learning with complex interplays among different features of various mobile users? To minimize the regret of decision-making under uncertain processing delays, adaptive learning that dynamically responds to hidden user features and uncertain processing delays should be devised, such that the obtained solution quickly converges to a near optimal solution [2].

To tackle the afore-mentioned challenges, in this paper we investigate the problem of service caching and task offloading with bursty data traffic and uncertain processing delays. It is worth mentioning that solutions to this problem will have wide applications in many edge scenarios, such as smart traffic and smart surveillance. For example, when there is a sudden traffic accident in a metropolitan area, a surge of user requests comes from the users in the congested vehicles, causing unexpected long network delays of the BS that is close to the accident location. By leveraging efficient and intelligent solutions, we could proactively handle the negative effects caused by bursty data traffic, by caching popular services to the location and forwarding the requests to BSs of the backhaul network.

To the best of our knowledge, we are the first to investigate both bursty data traffic and uncertain processing delays for the dynamic service caching and task offloading problem in an MEC network [5,8–18]. Besides, we are also the first to jointly consider a time-varying MEC network with both cellular networks and edge backhaul network and uncertain data traffic of users in the network; instead, existing studies dealt with task offloading and service caching without considering uncertain data traffic or in a single type of network [12,19,20]. Further, we are the first to devise a novel structural learning method, based on Generative-Adversarial Network (GAN) guided prediction algorithm, by adopting InfoGAN and using mutual information to accurately predict time series data, with the aim of avoiding model overfitting and collapse. The main contributions of this paper are summarized as follows.

- We propose an online learning algorithm for the joint service caching and task offloading problem with uncertain processing delays, based on the Multi-Armed Bandit (MAB) method. We also bound the regret of the proposed algorithm analytically.
- We propose an adaptive online learning algorithm with a bounded regret for the problem with both bursty data traffic and uncertain processing delays.
- We devise a GAN-guided prediction method to accurately predict the user demands based on small amounts of historical information. Based on the proposed GAN model, we propose a learning-aided heuristic for the service caching and task offloading problem with both bursty data traffic and uncertain processing delays.
- We evaluate the performance of the proposed online learning algorithms through experimental simulations, using real data traces. Experimental results show that the performance of the proposed algorithms outperforms existing ones.

The remainder of the paper is arranged as follows. Section 2 reviews related works on joint service caching and task offloading, and distinguishes this work from previous studies. Section 3 introduces the system model, notations, and problem formulation. Section 4 proposes an online learning algorithm to dynamically cache services and offload tasks in an MEC network. Section 5 devises an online learning algorithm for dynamic service caching and task offloading problem with bursty data traffic and uncertain processing delays. Section 6 devises a GAN-guided method to accurately predict the user demands and a heuristic based on the proposed novel GAN model. Section 7 evaluates the performance of the proposed algorithm by extensive simulations. Section 8 concludes the paper, and discusses future works of this paper.

2. Related work

The technique of MEC is emerging as a key enabling technique to address limited computing power of mobile devices and long service delays [8–10,21–23]. Most existing studies in literature focused on either task offloading or service placement in MEC networks. For instance, Chen et al. [24] formulated the multi-user computation offloading problem in a multi-channel wireless interference environment and proposed a game theoretic method. Chen et al. [25] proposed a joint mobility-aware caching and BS placement scheme. Huang et al. [26] studied the problem of reliable service placement in an MEC. Li et al. [27] addressed the problem of task offloading with statistical Quality of Service (QoS) guarantee in an MEC network. Jia et al. [28] proposed an online task offloading algorithm for both a sequence of tasks or concurrent tasks, by considering both heterogeneous tasks and dynamic channel status. Liang et al. [1] studied the problem of utility-based service caching, to minimize the cache cost while meeting the minimum utility requirement of each client, and they proposed an approximation algorithm with an approximate ratio for the problem. Badri et al. [29] studied a mobility-aware application placement problem, with an objective to maximize the QoS of the system while meeting the energy budget constraint of users. Xia et al. [30] investigated the service caching problem for delay-sensitive applications in an MEC network, by considering QoS requirements and budget constraints on user devices. Xiao et al. [23] focused on the security problem of task offloading in an MEC network, by applying reinforcement learning (RL) techniques to provide secure offloading to the edge nodes against jamming attacks. Novel lightweight authentication and secure collaborative caching schemes to protect data privacy are also proposed. Xiao et al. [31] proposed a new RL based task offloading scheme in MEC networks against jamming attacks and interference. Hu et al. [32]

considered the problem of offloading with uncertain task execution times, by proposing a learning-based scheme to predict the task execution times before offloading the tasks based on the relation between execution times and server configurations. Cheng et al. [33] studied the problem of task offloading in a 6G MEC network, where the tasks are offloaded into unmanned aerial vehicles (UAVs) for processing. Wang et al. [34] proposed a method to model the task arrivals in an MEC system by considering time-correlated traffics and network conditions. Fan et al. [19] investigated the problem of task offloading in an MEC network with a single edge server, considering uncertain computing delays and deterministic data volumes. Wang et al. [20] proposed a distributed algorithm for the problem of task offloading with uncertain computing and transmission delays. None of these studies focused on jointly considering service caching and task offloading in MEC networks with bursty data traffic and uncertain processing delays.

The studies closely related to this study are [5,11,14–16], in which they considered joint service caching and task offloading. For instance, Liu et al. [14] investigated the problem of joint offloading and caching in wireless blockchain networks. Xu et al. [11] devised an online algorithm for the problem of joint service caching and task offloading in a dense cellular network. Zhang et al. [18] studied the problem of collaborative task offloading and data caching, where users share data contents requested for computation tasks. Yan et al. [35] proposed an MEC service pricing scheme for wireless devices to coordinate service caching decisions and wireless devices' task offloading behaviors in a cellular network. Zhao et al. [36] studied how to efficiently offload dependent tasks to edge nodes with order-dependent and predetermined service caching. In addition, there are researches focusing on tasks offloading in an MEC network, by considering user mobility [37] or variety of network structure [38,39]. For example, Yang et al. [40] introduced an architecture that allows one user to be associated with multiple BSs, and investigated the interference of links together with the various computational capacity of BSs. Li et al. [41] investigated the joint online task offloading and service placement in dense MEC networks, with the aim to maximize the long-term average network utility while maintaining the stability of the MEC network. These studies however do not consider the bursty data traffic and uncertain processing delays of BSs in the MEC network.

There are also several studies addressing bursty data traffic or different uncertainties of MEC networks. Most of them however did not address the problem of joint service caching and task offloading. For instance, Apostolopoulos et al. [42] investigated the problem of task offloading in an MEC network by considering computation uncertainty and communication uncertainty. Chen et al. [13] devised a deep-reinforcement-learning-based scheduler to handle bursty requests in the process of task offloading, by assuming that network delays and user demands are given in advance. Hu et al. [43] studied the task offloading problem with partially given task execution time in an MEC network, and proposed a learning-driven offloading algorithm that learns task execution times before offloading. Chen et al. [44] proposed a recurrent neural network based prediction algorithm to learn high-dimensional and highly-variable cloud workloads. Niu et al. [45] proposed an approach to rapidly mitigate load imbalance due to unexpected bursty traffic. Zhao et al. [46] designed an edge server placement and server scheduling policy under workload uncertainty for 5G networks. Deng et al. [47] introduced a strategy to balance the service load when the load suddenly increases.

This paper is an extension of a conference paper [48]. The main differences between this paper and the conference version are as follows. We revise the delay model of requests to better fit real-world systems. We give the accurate definition of *the dynamic service caching problem with both bursty data traffic and uncertain processing delays*, in the form of integer linear program. We also devise two mechanisms for the problem and analyzed their regrets. We re-conduct all experiments to include the performance of the newly proposed mechanisms for the reformulated service caching and task offloading problem, in addition to the experimental results in the conference paper.

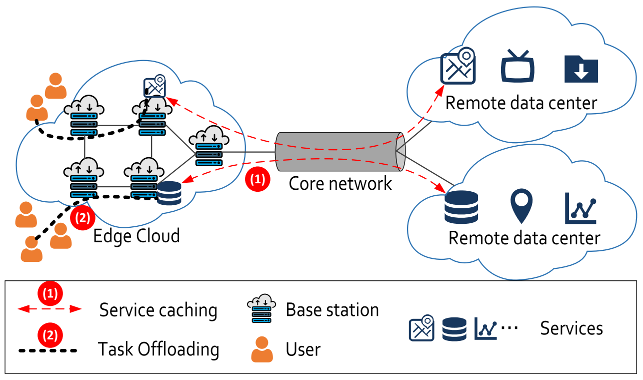


Fig. 1. An example of an MEC network. The service caching procedure in the MEC network usually consists of two steps (1) instances of different types of services are firstly cached into base stations and instantiated for the processing of user requests; (2) user requests are offloading to the cached instances of their required types of services.

3. Preliminary

In this section, we first introduce the system model, request and delay models. We then define the service caching problem with both bursty data traffic and uncertain processing delays precisely.

3.1. System model

We consider an MEC network $G = (\mathcal{BS}, E)$ operated by a network service provider, with a set $\mathcal{BS}$ of BSs providing computing resource for mobile users and a set E of backhaul links/paths that interconnect the BSs and a remote data center $\mathcal{DC}$ in the core network. Each BS $bs_i \in \mathcal{BS}$ provides wireless communication for mobile users, and it is attached with a certain amount of computing resource, such as an edge server or an AI accelerator such as Intel Movidius Neural Compute Stick [49]. As such, bs_i has computing capacity for executing tasks of mobile users, which is denoted by $C(bs_i)$. Note that the computing capacity of bs_i can be virtualized through cloud technologies, such that network services and user tasks can be executed in Virtual Machines (VMs) or containers. Each BS bs_i covers a set of users within its transmission range via operating over the orthogonal full-duplex spectrum. The transmission ranges of different BSs may overlap, since each BS is assigned a licensed band that may overlap with the bands of other BSs. Fig. 1 shows an example of the MEC network. The overall procedure for the service caching and task offloading model comprises the following steps: (1) **Request arrival:** user requests with bursty data traffic arrive at the system initially; (2) **Service caching:** each BS dynamically caches and initializes services when the data traffic of user requests changes; (3) **Task offloading:** users continuously send request data, which is dynamically routed to the assigned BS for processing by the cached service. The detailed descriptions are provided below.

3.2. User requests with bursty data traffic

Mobile users access the MEC network for various services, such as Virtual Reality (VR), cloud gaming, and IoT data processing, by connecting to their nearby BSs. Specifically, each mobile user sends a request with a certain amount of data traffic to be processed by its service. Assuming that a finite time horizon T is divided into equal time slots, each mobile user continuously sends requests with different amounts of data traffic for processing in each time slot t of the finite time horizon T. A set R of requests arrive in the system at the beginning of the finite time horizon T. Let r_l be such a request in R that demands a service $S(r_l)$ in the time horizon T. Note that each request has continuous data traffic during the finite time horizon T; however, it may have different volumes of data traffic in each time slot of T.

Therefore, the data traffic of each request r_l usually has a bursty pattern in different time slots of T. For example, VR services of a museum may experience a bursty amount of inference data when there is a significant surge of visitors in a certain period. However, the data traffic of each request is not fully unpredictable. By analyzing the recent running logs of a service, we can at least obtain the minimum amount of data traffic of each request. We thus assume that the minimum data traffic of a request within finite time horizon T is given as a priori. For clarity, we define the minimum data traffic of each request r_l as its *basic data traffic*, denoted by ρ_l^{bsc}. Note that the basic data volume usually reflects the basic behavior of a service, and may not change in the past few time horizons.

Besides the basic data traffic, each request can have a surge of its data traffic, referred to as *bursty data traffic*. The bursty data traffic of r_l basically is uncertain and not known in advance. Denote by $\rho_l^{bst}(t)$ the bursty data traffic of r_l in time slot t of the finite time horizon T. Such bursty data traffic is usually higher than the basic data traffic, i.e., $\rho_l^{bst}(t) \geq \rho_l^{bsc}$. Also, $\rho_l^{bst}(t)$ is uncertain given the unexpected nature of events with the bursty data traffic. Let $\rho_l(t)$ be the amount of data that needs to be processed for request r_l in time slot t. The data traffic $\rho_l(t)$ of r_l at time slot t can be either $\rho_l^{bst}(t)$ if it has a bursty pattern; otherwise $\rho_l(t) = \rho_l^{bsc}$. Each request r_l can be represented by $\langle \rho_l(t), S(r_l) \rangle$. Without loss of generality, to guarantee fair resource usage and service performance, we assume that each unit amount of data is assigned to an amount δ_{unit} of computing resource. Note that this is common assumption in many existing studies on resource allocation in MEC networks [50–52]. As such, requests with higher data traffic will be assigned to a higher amount of computing resource. The computing resource demand of request r_l in time slot t thus is $\delta_{unit} \cdot \rho_l(t)$.

3.3. Service caching

We focus on resource-hungry services that are originally deployed in remote data centers of the MEC network, such as services for VR applications. We assume that such services can be migrated to BSs of the MEC network. For example, a service that is implemented as a TensorFlow service running in a remote data center can also be simply migrated to the MEC network using TensorFlow Lite [53]. The migration of a service from remote data centers to a BS in an MEC network is referred to as *service caching*. At the beginning of each time slot $t \in T$, services migrate from remote data centers or other BSs for serving bursty data traffic of user requests, ensuring sufficient computing resources and minimal delay. Once a service is cached into a BS, it will be instantiated in the BS, which is referred to as an *instance* of the service. Note that multiple instances of a service can be cached into different BSs of the MEC network to serve requests in different locations. Without loss of generality, we assume that there is a single instance of a service in each BS, and the computing resource of the instance can be scaled according to the requests assigned to it. The cached instances of the service can serve requests within their proximity and handle the bursty data traffic of users timely. If a service instance is no longer used by requests, the service instance will be removed from its cached BS. Denote by $\mathcal{S}$ a set of to-be-cached services, and let $S_k \in \mathcal{S}$ be a service in $\mathcal{S}$ with $1 \leq k \leq |\mathcal{S}|$. Let R be the set of requests that need to be implemented by services in $\mathcal{S}$ in the finite time horizon T. Each service $S_k \in \mathcal{S}$ may be demanded by multiple user requests. The data traffic $\rho_l(t)$ of each request $r_l \in R$ needs to be transferred to a cached instance of its required service $S(r_l)$ for processing.

3.4. Uncertain processing delays

The delay experienced by a request in different cached service instances varies significantly, since it depends on many uncertain factors, such as bursty data traffic, user locations, and the congestion levels of BSs. On one hand, the data traffic of each request is bursty and uncertain. As such, the amount of data traffic waiting for processing by S_k may not be stable, leading to time-varying congestion levels of the BS hosting S_k. On the other hand, the processing rate of service S_k may not be stable as well, since its processing rate depends on the capability of its hosting containers or VMs that could be scaled in/out by the MEC network. Consequently, the processing delay experienced by each request is uncertain, which is referred to as *uncertain processing delays*. Even though, we can still get the maximum and minimum values of implementing a request r_l in a BS. Denote by $d_{li}(t)$ the delay experienced by request r_l if its data traffic is processed by an instance of service $S(r_l)$ in BS bs_i in time slot t. Note that the distribution of such delays is usually not known due to the complex interplay between various factors. Therefore, the mean and variance of the processing delays are not known in advance. For a single type of service (basically a piece of software), its code and initial data needed to run an instance is fixed as its software is compiled. However, the speed of loading the service (its code and data) varies from base station to base station. The reason is that base stations may have different computing capabilities and are located in different network locations. As such, the loading of code and data leads to different delays. This is the reason that we consider the instantiation delay is determined for each service and base station. Given the fact that base stations have relatively limited vendors with different computing capacities, we thus assume that the instantiation delay of caching an instance of service S_k in base station bs_i can be obtained from historical runtime operational data and given as a priori, denoted by d_{ik}^{ins}.

3.5. Problem definitions

We consider an MEC network G with bursty data traffic of requests and uncertain processing delays of BSs, a set of services $\mathcal{S} = \{S_k \mid 1 \leq k \leq |\mathcal{S}|\}$ to be cached, a set R of requests that are available at the beginning of the finite time horizon T, and each request $r_l \in R$ requiring to process its bursty data traffic in each time slot during the finite time horizon T. Note that although we assume that each request arrives into the system at the beginning of the finite time horizon, it may have bursty data traffic in each time slot of the finite time horizon. Moreover, given the network dynamics of the MEC network, the processing delays of each BS may be uncertain in each time slot of T. As such, requests may be implemented in different BSs of the MEC network during the finite time horizon T. The optimization problems considered in this paper are defined in the following.

Problem 1. Assuming that the processing delays of BSs are uncertain and the data traffic of each request in each time slot of the finite time horizon T is given, *the dynamic service caching problem with uncertain processing delays* aims to minimize the average delay experienced by each request, subject to the computing resource capacity constraint of each BS of the MEC network. To this end, we need to dynamically cache new instances of each service in $\mathcal{S}$ into BSs of the MEC network and assign each request r_l to one of the already or newly cached instances of its service $S(r_l)$ in each time slot t of the finite time horizon T. Note that we assume that the accumulative resources of all BSs are no less than the total resource demand of all requests; however, the resources that are available in a single BS may not be sufficient for the demand of all requests. Without loss of generality, we further assume that each request consumes at most α percentage computing resource of each BS, where $0 < \alpha \leq 1$. Note that this assumption comes from realistic scenarios where each BS is built to meet most of the user demands in its coverage, and each network service provider may pose a threshold on the maximum resource that can be assigned to each request for the sake of fairness in implementing other requests. Then, we have $\rho_l(t) \cdot \delta_{unit} \leq \alpha C(bs_i)$ for each BS $bs_i \in \mathcal{BS}$.

Problem 2. In real scenarios, data traffic of requests is bursty and processing delays of BSs are uncertain in each time slot of the finite time horizon T. *The dynamic service caching problem with both bursty data traffic and uncertain processing delays* in a finite time horizon T is to dynamically cache new instances of each service in $\mathcal{S}$ into BSs of the MEC network G and assign each request r_l to one of the already or newly cached instances of its required service $\mathcal{S}(r_l)$ in each time slot t of T. The objective of the problem is to minimize the average delay experienced by each request, subject to the computing resource capacity constraint of BSs.

The relations of the above-mentioned optimization problems are in the following. Although they share some similarities, they are abstracted from different application scenarios in an MEC network. The dynamic service caching problem with uncertain processing delays basically deals with MECs with dynamics on the network side, by assuming that user requests may have predicted or stable data traffic. For instance, the user requests from commuters on metro stations may have predicted data traffic on the same time period of a day. However, the dynamic service caching problem with both bursty data traffic and uncertain processing delays considers an additional dynamics from user requests. For instance, VR services of a museum may experience a bursty amount of inference data when there is a significant surge of visitors in a certain period, caching services for such VR services can be classified as the second problem.

We aim to design efficient and effective online learning algorithms with bounded regret for the defined optimization problems. It must be mentioned that regret bound is a common metric in the analysis of online learning algorithms. In other words, a regret bound measures the performance of an online algorithm relative to the performance of a competing adversary that knows any uncertain information of the system. Considering that the optimization objective is to minimize the average delay of each request, the regret we consider is in terms of the average delay of implemented requests. However, the regret concept can be easily extended to other metrics. For example, although the replacement of cached service instances is out of the scope of this paper, we can easily use the concept of regret to consider the cache hit ratio, by assuming both the algorithms and the competing adversary aim to optimize the cache replacement procedure to improve cache hit ratio of cached service instances.

The symbols used in this paper and their meanings are summarized in Table 1.

Table 1
Symbols.

Symbols	Meaning
$G = (\mathcal{BS}, E)$	An MEC network with a set $\mathcal{BS}$ of BSs and a set E of links
bs_i, $C(bs_i)$	A BS in $\mathcal{BS}$ and its computing capacity
T, t	A finite time horizon that is equally divided into time slots and a time slot t of T
r_l, $\mathcal{S}(r_l)$	A user request and its required service
$\mathcal{S}$, $\mathcal{S}_k$	A set of services and the service of type k
R	A set of requests of all time slots in a finite time horizon
ρ_l^{bsc}, $\rho_l^{bst}(t)$	The basic data traffic of request r_l and its bursty data traffic in time slot t
$\rho_l(t)$	The amount of data that needs to be processed for request r_l in time slot t
ϕ_l	The mean of the bursty data traffic during the finite time horizon T.
δ_{unit}	The amount of computing resource that is assigned to process a unit amount of data
$d_{li}(t)$	The delay experienced by request r_l if its data traffic is processed by an instance of service $\mathcal{S}_k$ in bs_i
d_{ik}^{ins}	The instantiation delay of caching an instance of service $\mathcal{S}_k$ in BS bs_i
$X_{li}(T)$, $\theta_{li}(t)$	The random process of the processing delay of a request r_l in bs_i during the finite time horizon T and its observed mean since time slot t
ϵ_t	The probability of exploring some random arms of the MAB process in time slot t
x_{lit}	A binary indicator variable that shows whether request is assigned to an instance of its service $\mathcal{S}_k$ in BS bs_i in time slot t
y_{kit}	A binary indicator variable that shows whether there is a cached instance of service $\mathcal{S}_k$ in BS bs_i in time slot t
x^*, y^*	The obtained fractional solution to **LP**
γ	A filtering threshold for x^* to make sure that BSs with higher probabilities are considered as *candidate BSs*
$\mathcal{BS}_{lt}^{candi}$	The candidate BS set for r_l in time slot t
$\mathcal{R}(t)$, $\mathcal{R}(T)$	The regret in time slot t and the regret in time horizon T
$x_{lit}^{opt}(\tau)$, $y_{kit}^{opt}(\tau)$	The optimal request assignment and service caching in time slot τ
h, h'	The number of time slots that an optimal BS is found and the number of time slots that the request is assigned to its optimal BS
x'_{li}	A binary indicator variable that shows whether request r_l is assigned to its service in BS bs_i in **ILP-UN**
y'_{ki}	Indicator variable showing whether an instance of $\mathcal{S}_k$ is cached into bs_i in **ILP-UN**
x'^*, y'^*	The obtained fractional solution to **LP-UN**
Z, Z'	The set of candidate values of the threshold γ and a subset of Z
$n_t(\gamma_c)$	The number of time slots before t when γ_c is chosen as the value of γ
$\psi_t(\gamma_c)$	The obtained average reward for γ_c
$r_t(\gamma_c)$ and $B_t(\gamma_c)$	The confidence radius if a value γ_c and Lipschitz condition
$index_t(\gamma_c)$	The index for a time slot t and for a value $\gamma_c \in Z$

4. Online learning algorithm for the dynamic service caching problem with uncertain processing delays

We now devise an online learning algorithm with a bounded regret for the dynamic service caching problem with uncertain processing delays, via a novel integration of a randomized rounding method with the MAB method.

4.1. Basic idea

To handle the uncertain processing delay experienced by each request in BS bs_i, we adopt the MAB method to learn the delay proactively. Specifically, we consider each combination of request and BS as an arm of a *bandit*. The processing delay of offloading request r_l to bs_i can be obtained only when the actual offloading happens; that is the corresponding arm of the combination of r_l and bs_i is played at the beginning of each time slot t. Let $X_{li}(T)$ be such a random process of the processing delay of request r_l in bs_i during the finite time horizon T. However, we can record the historical processing delays of playing each arm before time slot t and obtain the mean processing delays of the time slots with each time slot t' with $1 \leq t' < t$. Let $\theta_{li}(t)$ be the mean of $X_{li}(T)$ that is obtained so far at the beginning of time slot t. Since we do not know the distribution of $X_{li}(T)$, we need to progressively learn the distribution of the processing delays by observing and updating the mean of $X_{li}(T)$ when an arm is played for request r_l. That is, at the end of each time slot t, we observe the real processing delays and obtain $\theta_{li}(t+1)$ for the next time slot $t+1$. When $t = 1$, $\theta_{li}(1)$ is set as a pre-defined fixed value. When $t > 1$, for the combinations of requests and BSs that have not been explored, $\theta_{li}(t)$ is set to the mean value of the explored requests in the same BS. Then, at the beginning of each time slot t, the algorithm makes decisions of service caching and task offloading in time slot t, based on the value of $\theta_{li}(t)$.

To find optimal or near-optimal solutions to the dynamic service caching problem in each time slot t, a feasible solution is to find an approximate solution that is not far from the optimal one based on the value of $\theta_{li}(t)$. Specifically, by solving the relaxed LP of the formulated ILP of the problem, we could obtain the probabilities of selecting each arm. However, since $\theta_{li}(t)$ only represent the mean processing delay of the time slots before time slot t, and does not represent the actual processing delays of the current time slot t, the obtained solution can deviate from the actual optimal solution. Therefore, this may make some actual optimal arms not being selected. To avoid this, we allow the algorithm to explore arms that may not have been suggested by the ILP while they may be the actual optimal arms with a probability that dynamically changes. In other words, we propose a novel online learning algorithm that merges an adaptive exploit-and-explore method and near-optimal solutions obtained from an exact solution. However, we need to find an optimal ϵ_t such that the regret of the obtained solution is bounded. To this end, we devise an approximation algorithm for the dynamic service caching problem in each time slot t such that the quality of the obtained solution is not far from the optimal one. The algorithm then randomly explores other arms that are not specified by the approximate solution with probability ϵ_t.

4.2. Online learning algorithm

We now propose an approximation algorithm with an approximation ratio for the dynamic service caching problem with uncertain processing delays in each time slot t of the finite time horizon T. The basic idea is to formulate the problem into an Integer Linear Program (ILP), based on the mean value of $\theta_{li}(t)$, and then based on a fractional solution of the ILP to select optimal arms in the MAB method.

Recall that the objective of the problem is to minimize the average delay experienced by each request, by dynamically caching an instance of each service in $\mathcal{S}$ and assigning each request r_l to a cached service instance in each time slot t. Denote by x_{lit} a binary variable that shows whether request r_l is assigned to an instance of its required service $\mathcal{S}(r_l)$ in BS bs_i in time slot t of the finite time horizon T. Let y_{kit} be a binary variable showing whether an instance of $\mathcal{S}_k$ is cached to BS bs_i in time slot t. Note that there may be multiple instances for service $\mathcal{S}_k$ in different BSs of the MEC network. The objective can be formulated as

$$\textbf{ILP(t)} : \min \frac{1}{|R|}\Big(\sum_{r_l\in R}\sum_{bs_i\in\mathcal{BS}} x_{lit}\cdot\theta_{li}(t)+\sum_{S_k\in S} y_{kit}\cdot d^{ins}_{i,k}\Big), \tag{1}$$

subject to the following constraints,

$$\sum_{bs_i\in\mathcal{BS}} x_{lit} = 1, \qquad \forall r_l \in R \tag{2}$$

$$\sum_{r_l\in R} x_{lit}\cdot\rho_l(t)\cdot\delta_{unit} \le C(bs_i), \quad \forall bs_i \in \mathcal{BS} \tag{3}$$

$$y_{kit} \ge x_{lit}, \qquad \forall r_l \in R \text{ and } \mathcal{S}_k = \mathcal{S}(r_l) \tag{4}$$

$$x_{lit}, y_{kit} \in \{0,1\}. \tag{5}$$

Constraint (2) requires that each request r_l must be served by a BS. Constraint (3) says that the total resource demand of implementing these requests in BS bs_i should not exceed its capacity. Constraint (4) means if request r_l is assigned to BS bs_i, its required service $\mathcal{S}(r_l)$ has to be cached into bs_i to process its data.

We can solve **ILP(t)** to obtain the optimal solution to the problem. To proactively respond to network dynamics, the length of a time slot needs to be as small as possible. However, a shorter length of time slot may not be enough to solve the **ILP(t)**, especially for large problem sizes. Instead, we relax the **ILP(t)** into a Linear Program (LP), by relaxing integer variables of the **ILP(t)** into

$$0 \le x_{lit}, y_{kit} \le 1. \tag{6}$$

We then solve the **LP(t)** and obtain a fractional solution in polynomial time. As shown in Fig. 2, we then select BSs (arms) according to this fractional solution. Let x^* and y^* be the obtained fractional solution. However, the fractional solution is not a feasible solution to the dynamic service caching problem, since each service or request cannot be 'split' and assigned to different BSs.

Our strategy is to select optimal arms in each time slot t according to the fractional solution obtained by **LP(t)**, to ensure the performance guarantee of the solution. Specifically, we consider each fractional value of the fractional solution as a probability. For example, a higher value for x^*_{lit} means a higher probability that the selection of BS bs_i incurs a better performance in terms of lower delays. Intuitively, we would like to select the arms with higher values of x^*_{lit}. Simply rounding a small value of x^*_{lit} to 1 may result in significant violations on the resource capacity of BSs. We thus define a threshold γ for each x^*_{lit} to make sure that BSs with higher probabilities are considered as *candidate BSs*. Denote by $\mathcal{BS}^{candi}_{lt}$ the set of BSs for request r_l in time slot t, which can be built by

$$\mathcal{BS}^{candi}_{lt} = \{bs_i \mid x^*_{lit} \ge \gamma\}. \tag{7}$$

Once the candidate set of BSs for each request r_l in time slot t is obtained, it is ideal to select a BS from its candidate set by treating each x^*_{lit} as the selection probability. Then, y_{kit} can be recalculated by examining each request assigned to BS bs_i. We can obtain the information of the selected BSs. Besides, since the processing delay of each BS significantly varies, we also make some explorations to use other BSs that are not in the candidate set. That is, we adopt an adaptive ϵ-greedy strategy, in which we choose an arm from the candidate set with a probability $\epsilon_t = 1/t$, or randomly pick an arm inside the whole $\mathcal{BS}$ set to explore better BSs for requests. By doing this, we exploit 'good BSs' that are considered having the potential to minimize the average processing delay of requests according to the probability of x^*_{lit}. In the meantime, we explore some other BSs. In particular, when the algorithm chooses to explore BSs, for the sake of fairness, we assume that each BS bs_i is selected with probability

$$\frac{C(bs_i)}{\sum_{bs_j\in\mathcal{BS}} C(bs_j)}. \tag{8}$$

The rationale behind is that the BSs with high computing resource capacities are selected with a high probability.

The detailed steps of the proposed algorithm for the dynamic service caching problem with uncertain processing delays are shown in **Algorithm 1**.

Algorithm 1 `OL_GD`

Input: $G = (\mathcal{BS}, E)$, a set of services $\mathcal{S}$, and a set of requests R.
Output: The caching of each service $S_k \in \mathcal{S}$ and the assignment of each request $r_k \in R$ to its cached service in a BS.

```
1: for $t \leftarrow 1, \cdots, T$ do
2:    $\epsilon_t \leftarrow \frac{1}{t}$;
3:    Relax the ILP(t) in (1) into an LP(t);
4:    Obtain a fractional solution to the LP(t) and get a candidate set $\mathcal{BS}^{candi}_{lt}$
      of BSs for each request $r_l$ in time slot $t$;
5:    Pick a random number in the range of $[0, 1]$;
6:    if the random number $< 1 - \epsilon_t$ then
7:       Assign each request $r_l$ to BS $bs_i$ in $\mathcal{BS}^{candi}_{lt}$ with probability $x^*_{lit}$;
8:    else
9:       Randomly select a BS $bs_i \in \mathcal{BS}$ with probability $C(bs_i)/\sum_{bs_j\in\mathcal{BS}} C(bs_j)$
         for each request $r_l$;
10:   end if
11:   for each BS $bs_i$ with assigned requests do
12:      Observe the processing delay and update its mean value $\theta_{li}(t+1)$ for
         each request $r_l$ in the next time slot $t+1$;
13:   end for
14: end for
```

4.3. Algorithm analysis

We now analyze the correctness and the performance of algorithm `OL_GD`. In the following, we first show the resource violation ratio of each BS and its probability in each time slot t of a finite time horizon T. We then analyze the regret of the online learning algorithm `OL_GD` during the finite time horizon T. We finally present the time complexity analysis of the algorithm `OL_GD`.

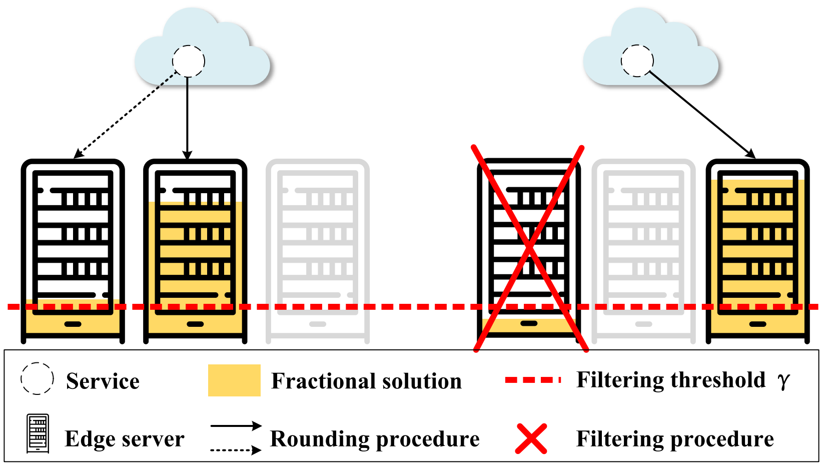


Fig. 2. The rounding and filtering procedure of algorithm OL_GD.

Lemma 1. *Assuming that the accumulative resources are no less than the total resource demands and* $\rho_l(t)\cdot\delta_{unit}\le\alpha C(bs_i)$*, the maximum probability of violating the computing resource capacity of any BS by* $(1+q)$ *times is* $\exp(-\frac{2q^2}{\alpha^2|R|})$*, where* α *is a given constant in the range of* $(0,1]$.

Proof. Denote by H_{it} the amount of computing resource that is used to process the requests that are assigned to BS bs_i in time slot t of the finite time horizon T. Clearly, we have

$$E(H_{it})=\sum_{r_l\in R}x^*_{lit}\cdot\rho_l(t)\cdot\delta_{unit}. \tag{9}$$

As shown in algorithm OL_GD, we know that a request r_l is assigned to a BS in $\mathcal{BS}^{candi}_{lt}$ with probability $1-1/t$. We thus consider the following two cases: **case 1:** request r_l is implemented according to the filtered fractional solution due to **LP(t)** with probability $1-1/t$, and it is assigned to a BS $bs_i\in\mathcal{BS}^{candi}_{lt}$ with probability x^*_{lit}; and **case 2:** the algorithm explores better BSs for r_l with probability $1/t$, and r_l is randomly assigned to a BS out of all BSs in $\mathcal{BS}$ with probability $C(bs_i)/\sum_{bs_j\in\mathcal{BS}}C(bs_j)$.

First, **case 1** happens with probability $1-1/t$. Besides, since the $\mathcal{BS}^{candi}_{lt}$ is obtained by solving the **ILP(t)**, constraint (3) is met in the fractional solution to the relaxed version of **ILP(t)**. Then, we have

$$E_1(H_{it})=(1-\frac{1}{t})\sum_{r_l\in R}x^*_{lit}\cdot\rho_l(t)\cdot\delta_{unit}\le(1-\frac{1}{t})C(bs_i). \tag{10}$$

For **case 2**, each randomly assigned request is assigned to bs_i with probability $C(bs_i)/\sum_{bs_j\in\mathcal{BS}}C(bs_j)$. Then, we have

$$\begin{aligned}E_2(H_{it})&=\frac{1}{t}\frac{C(bs_i)}{\sum_{bs_j\in\mathcal{BS}}C(bs_j)}\sum_{r_l\in R}\rho_l(t)\cdot\delta_{unit}\\&\le\frac{1}{t}\frac{C(bs_i)}{\sum_{bs_j\in\mathcal{BS}}C(bs_j)}\sum_{bs_j\in\mathcal{BS}}C(bs_j),\\&\qquad\text{since the accumulative resources is no less}\\&\qquad\text{than the total resource demands}\\&=\frac{1}{t}C(bs_i).\end{aligned} \tag{11}$$

The expectation of H_{it} thus can be expressed by

$$E(H_{it})=E_1(H_{it})+E_2(H_{it})\le C(bs_i). \tag{12}$$

Let $Pr[\cdot]$ be the probability of an event. The computing resource constraint of BS bs_i is violated by $(1+q)$ times if the total computing resource of the remaining requests in bs_i is still at least $(1+q)C(bs_i)$ after taking out an assigned request. To calculate such probability is to show

$$Pr[H_{it}\ge(1+q)C(bs_i)].$$

We then can calculate the violation probability by (see [54])

$$\begin{aligned}&Pr[H_{it}\ge(1+q)C(bs_i)]\\=&Pr[H_{it}-E(H_{it})\ge(1+q)C(bs_i)-E(H_{it})]\\\le&Pr[H_{it}-E(H_{it})\ge qC(bs_i)]\\\le&\exp(-\frac{2q^2(C(bs_i))^2}{\sum_{r_l\in R;\ x^*_{lit}\ge\gamma}(\rho_l(t)\cdot\delta_{unit}-0)^2}),\quad\text{following the Hoeffding bound}\\\le&\exp(-\frac{2q^2(C(bs_i))^2}{|R|(\alpha C(bs_i))^2}),\quad\text{since }\rho_l(t)\cdot\delta_{unit}\le\alpha C(bs_i)\\=&\exp(-\frac{2q^2}{\alpha^2|R|}).\end{aligned} \tag{13}$$

The theorem holds.

We then define the regret of the online learning algorithm in time slot t as the difference between the expected delay obtained by the proposed algorithm and that could be obtained by picking an optimal BS at all previous time slots with $\tau\le t$. The regret in time slot t thus can be expressed by

$$\begin{aligned}\mathcal{R}(t)=&E[\sum_{r_l\in R}\sum_{bs_i\in\mathcal{BS}}x^*_{lit}\cdot X_{li}(T)+y^*_{kit}\cdot d^{ins}_{ik}]\\&-(\sum_{r_l\in R}\sum_{bs_i\in\mathcal{BS}}x^{opt}_{lit}\cdot d_{li}(t)+y^{opt}_{kit}\cdot d^{ins}_{ik})\\\le&E[\sum_{r_l\in R}\sum_{bs_i\in\mathcal{BS}}x_{lit}\cdot X_{li}(T)+y_{kit}\cdot d^{ins}_{ik}]\\&-\min_{1\le\tau\le t}\{\sum_{r_l\in R}\sum_{bs_i\in\mathcal{BS}}x^{opt}_{li\tau}\cdot d_{li}(\tau)+y^{opt}_{ki\tau}\cdot d^{ins}_{ik}\},\end{aligned} \tag{14}$$

where $x^{opt}_{li\tau}$ and $y^{opt}_{ki\tau}$ are the optimal request assignment and service caching in time slot τ for any τ with $1\le\tau\le t$. The total regret in time horizon T can be expressed by

$$\mathcal{R}(T)=\sum_{1\le t\le T}\mathcal{R}(t). \tag{15}$$

To show the bound of regret of the proposed algorithm, we analyze the gap between the optimal service caching and the worst service caching in the following lemma.

Lemma 2. *Assuming that the delay* $d_{li}(t)$ *processing request* r_l *in BS* bs_i *in time slot* t *is not given while its maximum and minimum values are known in advance, the gap between the optimal service caching and the worst service caching is bounded by* $T\cdot|R|\max\{(d^{max}-\gamma\cdot d^{min}+\Delta^{ins}),\gamma\cdot e^{-2\gamma|R|^2}+\Delta^{ins}\}$*, where* $d^{max}=\max_{l,i,t}d_{li}(t)$*,* $d^{min}=\min_{l,i,t}d_{li}(t)$*, and* $\Delta^{ins}=\max_{i,k}d^{ins}_{ik}-\min_{i,k}d^{ins}_{ik}$.

Proof. From the definition of regret in Eq. (15), the regret can be considered in the following two cases: **case 1:** the regret due to the deviation from the optimal solution if the processing delay for request r_l in BS bs_i happens to be its currently observed mean value $\theta_{li}(t)$; and **case 2:** the regret due to the fact that the processing delay of request r_l in each BS bs_i deviates from $\theta_{li}(t)$.

Case 1. Let R' be the set of requests whose processing delays happen to be the observed mean value $\theta_{li}(t)$ of BS. Deriving the regret bound is equivalent to calculate the worst bound of

$$E(\mathcal{R}(T)) = E[\sum_{1\le t\le T}\sum_{r_l\in R'}\sum_{bs_i\in\mathcal{BS}} x^*_{lit}\cdot d_{li}(t) + y^*_{kit}\cdot d^{ins}_{ik}] - \sum_{1\le t\le T}\sum_{r_l\in R'}\sum_{bs_i\in\mathcal{BS}}\left(x^{opt}_{lit}\cdot d_{li}(t) + y^{opt}_{kit}\cdot d^{ins}_{ik}\right).$$

Recall that in algorithm OL_GD, BS bs_i is a candidate BS for r_l if $x^*_{lit} \ge \gamma$. In the worst case, a BS with the longest processing delay is selected with high probability, while the BS with the lowest processing is selected with the lowest probability γ. This means that

$$E(\mathcal{R}(T)) \le \sum_{1\le t\le T}\sum_{r_l\in R'}\sum_{bs_i\in\mathcal{BS}}\left(d^{max} - \gamma\cdot d^{min} + \Delta^{ins}\right) \le T\cdot|R'|\cdot(d^{max} - \gamma\cdot d^{min} + \Delta^{ins}). \quad (16)$$

Case 2. Denote by R'' the set of requests whose processing delays deviate from the mean processing delay of BSs. We can calculate the regret by

$$E(\mathcal{R}(T)) = E\big[\sum_{1\le t\le T}\sum_{r_l\in R''}\sum_{bs_i\in\mathcal{BS}}\theta_{li}(t) - \gamma\cdot d_{li}(t) + \Delta^{ins}\big].$$

By Chernoff–Hoeffding bound [54], we know that

$$Pr[d_{li}(t) > n\theta_{li}(t) - a] \le e^{-2a^2/n}, \quad (17)$$

which means that

$$Pr[\theta_{li}(t) - d_{li}(t)\cdot\gamma < |R''|\cdot\gamma] \le e^{-2\gamma|R''|^2}, \quad (18)$$

if $n = \frac{1}{\gamma}$ and $a = |R''|$.

We thus have

$$E(\mathcal{R}(T)) = E\big[\sum_{1\le t\le T}\sum_{r_l\in R''}\sum_{bs_i\in\mathcal{BS}}\theta_{li}(t) - \gamma\cdot d_{li}(t) + \Delta^{ins}\big] \le T\cdot|R''|(\gamma\cdot e^{-2\gamma|R|^2} + \Delta^{ins}). \quad (19)$$

Denote by σ the gap between the optimal service caching and the worst service caching. By jointly considering **case 1** and **case 2**, we have

$$\sigma = \max\{T\cdot|R'|(d^{max} - \gamma\cdot d^{min} + \Delta^{ins}), T\cdot|R''|(\gamma\cdot e^{-2\gamma|R|^2} + \Delta^{ins})\} \le T\cdot|R|\max\{(d^{max} - \gamma\cdot d^{min} + \Delta^{ins}), (\gamma\cdot e^{-2\gamma|R|^2} + \Delta^{ins})\} \quad (20)$$

The lemma holds.

We then show the bound on regret and the time complexity of algorithm OL_GD in the following theorem.

Theorem 1. *Given an MEC network $G = (\mathcal{BS}, E)$, a set R of requests, and a set $\mathcal{S}$ of services, assuming that $C(bs_i)/\sum_{bs_j\in BS} C(bs_j) \le 1-\gamma$, there is an online algorithm to the dynamic service caching problem with uncertain processing delays that has a bounded regret of $\sigma \log\frac{T-1}{e^{1/c}+1}$, when the proposed algorithm runs T time slots in a finite time horizon T. The time complexity of algorithm* OL_GD *is $O((|\mathcal{BS}|(|R|+|\mathcal{S}|))^{\omega}B)$ for a single time slot $t\in T$, where ω is the exponent of matrix multiplication and the current value of $\omega\approx 2.37$, and B is the input bits [55].*

Proof. We first show the regret bound of algorithm OL_GD. The regret can be divided into three cases: **case 1:** the regret due to the exploring of sub-optimal solutions that are not considered as candidate BSs; **case 2:** the regret because of exploring to discover the optimal solution, and **case 3:** the regret due to deviating from the optimal solution by having certain probability to explore suboptimal solutions.

Case 1. The sub-optimal solutions without the candidate set is explored with probability $(1-\gamma)$. Such BSs may be optimal solutions given their delay information. Let h be the number of time slots that an optimal BS is found, we can calculate h by

$$\sum_{t=1}^{h}(1-\gamma) \ge 1. \quad (21)$$

We thus have

$$h = 1/(1-\gamma). \quad (22)$$

Case 2. We can see that the probability of exploring solutions is $\frac{c}{t}$ in each time slot t, where $0 < c < 1$. Assuming that h' is the number of time slots that the request is assigned to its optimal BS, h' can be derived by

$$\sum_{t=1}^{h'}(\frac{c}{t}) \ge 1, \quad (23)$$

We then have

$$h' = e^{1/c}. \quad (24)$$

Assuming that $\gamma < 1 - e^{-1/c}$, the regret in parts (1) and (2) is

$$\sigma\cdot e^{1/c}. \quad (25)$$

Case 3. The expected regret till time slot t is

$$E(R(T)) \le \sigma\sum_{t=h+1}^{T}\frac{c}{t} \le \sigma\int_{t=e^{1/c}+1}^{T-1}\frac{1}{x}dx = \sigma\log\frac{T-1}{e^{1/c}+1}. \quad (26)$$

We then analyze the time complexity of algorithm OL_GD. The assignment in a single time slot t primarily involves two time-consuming parts: First, the solving of the **LP(t)**, and second, the running of the filtering and MAB-based rounding procedure. Using Lee and Song's algorithm [55], **LP(t)** solving procedure can take $O(g^{\omega}B)$ where $g = |\mathcal{BS}|(|R|+|\mathcal{S}|)$ is the number of decision variables, ω is the exponent of matrix multiplication, and B is the input bits. The filtering and MAB-based rounding are each applied once to every request r_l, resulting in an overall time complexity of $O(|R|)$ in a single time slot t. For the current value of $\omega\approx 2.37$, the solution to the **LP(t)** is more time-consuming. So the time complexity of OL_GD thus is $O((|\mathcal{BS}|(|R|+|\mathcal{S}|))^{\omega}B)$.

The theorem holds.

5. Online learning algorithm for the dynamic service caching problem with bursty data traffic and uncertain processing delays

We now propose an online learning algorithm for the dynamic service caching problem with both bursty data traffic and uncertain processing delays.

5.1. An online learning algorithm

The bursty data traffic of each request r_l is due to various factors of the MEC network. In the time slot t of the finite time horizon T, the bursty data traffic $\rho^{bst}_l(t)$ of request r_l is usually unknown before its implementation in the MEC network. However, the distribution of $\rho^{bst}_l(t)$ of a request is usually known and can be obtained through historical information of similar types of requests. Denote by ϕ_l the mean of the bursty data traffic during the finite time horizon T. A simple idea to address the challenges brought by bursty data traffic and uncertain processing delays is to extend the proposed algorithm OL_GD that ignores the bursty data traffic of requests. Specifically, we replace the actual data rate $\rho_l(t)$ in ILP with ϕ_l. Furthermore, we replace $\theta_{li}(t)$ in ILP with θ_{li} that denotes the mean delay of processing request r_l in BS bs_i in time slot t, i.e., $\theta_{li} = E[X_{li}(t)]$. Let **ILP-UN** be the obtained ILP. Denote by x'_{li} a binary indicator variable that shows whether request r_l is assigned to its service in BS bs_i in **ILP-UN**. y'_{ki} represents the indicator variable showing whether an instance of S_k is cached into bs_i. **ILP-UN** can be formulated as

$$\textbf{ILP}-\textbf{UN}: \ \min \ \frac{1}{|R|}\Big(\sum_{r_l\in R}\sum_{bs_i\in\mathcal{BS}} x'_{li}\cdot\theta_{li} + \sum_{S_k\in\mathcal{S}} y'_{ki}\cdot d^{ins}_{ik}\Big), \tag{27}$$

subject to the following constraints.

$$\sum_{bs_i\in\mathcal{BS}} x'_{li} = 1, \qquad \forall r_l\in R \tag{28}$$

$$\sum_{r_l\in R} x'_{li}\cdot\phi_l\cdot\delta_{unit} \le C(bs_i), \quad \forall bs_i \tag{29}$$

$$y'_{ki} \ge x'_{li}, \qquad \forall r_l\in R \text{ and its service } S_k \tag{30}$$

$$x'_{li}, y'_{ki} \in \{0,1\}. \tag{31}$$

Similar to algorithm OL_GD, we can relax Constraint (31) by solving the relaxed **ILP-UN**, and we obtain a candidate set of BSs for each request. It must be mentioned that in **ILP-UN**, we use the expected data volume of each request. We thus do not need to solve the **ILP-UN** at each time slot. Instead, in the proposed algorithm, we obtain fractional solutions x'^{*}_{li} and y'^{*}_{ki} at the very beginning, according to the given distributions of both data volumes and network delays. Then, in each time slot, we adopt an exploit–explore strategy similar to that used in algorithm OL_GD. The detailed algorithm is referred to as algorithm OL_UN, shown in algorithm 2.

Algorithm 2 OL_UN

```
Input: $G = (\mathcal{BS}, E)$, a set of services $\mathcal{S}$, and a set of requests $R$.
Output: The caching of each service $S_k \in \mathcal{S}$ and the assignment of each
    request $r_k \in R$ to its cached service in a BS.
1: Relax the ILP-UN in (27) into LP-UN;
2: Obtain a fractional solution to the LP-UN and get a candidate set $\mathcal{BS}^{candi}_l$
   of BSs for each request $r_l$;
3: for $t \leftarrow 1, \cdots, T$ do
4:     $\epsilon_t \leftarrow \frac{1}{t}$;
5:     Pick a random number in the range of $[0, 1]$;
6:     if the random number $< 1 - \epsilon_t$ then
7:         Assign each request $r_l$ to BS $bs_i$ in $\mathcal{BS}^{candi}_l$ with probability $x'^{*}_{li}$;
8:     else
9:         Randomly select a BS $bs_i \in \mathcal{BS}$ with probability $C(bs_i)/\sum_{bs_j\in\mathcal{BS}} C(bs_j)$
           for each request $r_l$;
10:    end if
11:    for each BS $bs_i$ with assigned requests do
12:        Observe the delay of processing request $r_l$ and update its mean value
           $\theta_{li}$;
13:    end for
14:    for each request $r_l$ do
15:        Observe the data traffic $\rho^{bst}_l(t)$ and update its mean value $\phi_l$;
16:    end for
17: end for
```

5.2. An adaptive learning algorithm

Algorithm OL_UN adopts a one-time assignment strategy according to the expected data traffic of each request and the expected processing delay of each BS. In each time slot, it solely relies on the exploit-and-explore procedure to find BSs with shorter processing delays. This may slow down the convergence of the algorithm if the data traffic of requests and processing delays vary in large ranges. The rationale is that the regret of the algorithm may be large if the actual data traffic and delays deviate from their expected values.

To avoid regret deviation, we now design an adaptive online learning algorithm via a fully-customized zooming algorithm [56]. Specifically, we determine the candidate set of BSs for each request r_l at the beginning of each time slot dynamically. With a little abuse of notations, we use $\mathcal{BS}^{candi}_{lt}$ to denote the set of candidate BSs for request r_l in time slot t. We observe the filtering threshold γ that determines the set of candidate BSs for each r_l. However, a predefined threshold sometimes can be too small that may result in large resource constraint violations, or too big that will filter out too many candidate locations for some specific requests for further assignment. Therefore, instead of using a fixed value for γ, we dynamically adjust threshold γ according to the current mean reward of each arm (a.k.a. BS). Given that the objective is to minimize the average delay experienced by each request, the reward is defined as the negative observed value of average delay after the requests have been assigned by the algorithm.

We assume that γ is a value in the range of $(0, 1 - e^{-1/c}]$ with c in the range of $0 < c < 1$, meaning that there are infinite settings for threshold γ. This makes the value selection for γ very challenging. To tackle this challenge, we here adopt an adaptive discretization method via proposing a new zooming algorithm. Specifically, we discretize the range of $[0, 1-e^{-1/c}]$ into κ intervals with fixed length $\xi = (1-e^{-1/c})/(\kappa-1)$, so that the obtained set Z for γ consists of all integer multiples of ξ. After the discretization of Z, a finite number of values in set Z is obtained.

To find a value for γ that achieves an optimal reward, we adopt an *activation and selection* mechanism that maintains a subset Z' ($\in Z$) of active values dynamically. Namely, some values in Z will be activated in each time slot and added into Z', and a value for γ will be selected from Z'. We adopt a similar strategy as the zooming algorithm [56]: a value will not be deactivated once its activated. In the following, we describe the activation and selection rules.

Activation rule: To decide which value in Z should be activated and put into set Z', we maintain an average reward for each value in Z' and define a confidence radius for the value. Let γ_c be such a value in Z'. The other values that can be covered by the radius of γ_c will not be activated. The rationale is that the values that are covered by the radius have similar average rewards, and it may not be necessary to activate other values if the reward difference is not large (i.e., within the confidence radius).

We now fix time slot t and a value $\gamma_c \in Z$. Denote by $n_t(\gamma_c)$ the number of time slots before t when γ_c is chosen as the value of γ, and let $\psi_t(\gamma_c)$ be the average reward obtained by the algorithm when $\gamma = \gamma_c$. The confidence radius, denoted by $r_t(\gamma_c)$, is defined by

$$r_t(\gamma_c) = \sqrt{2\log T/(n_t(\gamma_c)+1)}. \tag{32}$$

The confidence ball then consists of all values in Z that are in the set

$$B_t(\gamma_c) = \{\gamma'_c \in Z \mid \mathcal{D}(\gamma_c, \gamma'_c) \le r_t(\gamma_c)\}, \tag{33}$$

where $\mathcal{D}(\gamma_c, \gamma'_c) \le r_t(\gamma_c)$ is a condition for the Lipschitz bandits, i.e., $\mathcal{D}(\gamma_c, \gamma'_c) = L|\gamma_c - \gamma'_c|$, and L is the Lipschitz constant to scaling.

In each time slot t, we guarantee that all the values in Z are represented by the active arms in Z'. As time goes, the confidence ball of each value γ_c shrinks, thereby making some values not covered. We then simply add those non-covered values active by adding them to Z'.

Selection rule: Given a set of active values in Z', we simply select the value with the highest index, which is defined by

$$index_t(\gamma_c) = \psi_t(\gamma_c) + 2r_t(\gamma_c). \tag{34}$$

where '+' is a way to trade off exploration and exploitation, and the factor of two is needed to emphasize exploration that encourages arms with low mean rewards not to be played too often.

Having obtained the value for threshold γ, we then determine the candidate BS for each request r_l, based on the fractional solution to **ILP-UN**, i.e.,

$$\mathcal{BS}^{candi}_{lt} = \{bs_i \mid x'^{*}_{li} \ge \gamma\}. \tag{35}$$

The rest is the same as algorithm OL_UN, and the details of the proposed algorithm, denoted by OL_ADA are shown in algorithm 3.

Algorithm 3 OL_ADA

```
Input: $G = (\mathcal{BS}, E)$, a set of services $\mathcal{S}$, and a set of requests $R$.
Output: The caching of each service $S_k \in \mathcal{S}$ and the assignment of each request $r_k \in R$ to its cached service in a BS.
1: Relax the ILP-UN in (27) into LP-UN;
2: Initialize the set $Z$ of discretized values of $\gamma$; initialize $Z' = \{\}$;
3: for $t \leftarrow 1, \cdots, T$ do
4:     Compute the confidence balls for each value in $Z'$ by Eq. (32), (33), and find out the values in $Z$ not covered by any confidence ball;
5:     Add the not covered values to $Z'$ one by one until all values in $Z$ are covered;
6:     $\gamma \leftarrow \arg\max_{\gamma_c \in Z'} index_t(\gamma_c)$;
7:     Obtain a fractional solution to the LP-UN and get a candidate set $\mathcal{BS}_{lt}^{candi}$ of BSs for each request $r_l$, with given filtering threshold $\gamma$;
8:     $\epsilon_t \leftarrow \frac{1}{t}$;
9:     Pick a random number in the range of $[0,1]$;
10:    if the random number $< 1 - \epsilon_t$ then
11:        Assign each request $r_l$ to BS $bs_i$ in $\mathcal{BS}_{lt}^{candi}$ with probability $x_{li}^{\prime *}$;
12:    else
13:        Randomly select a BS $bs_i \in \mathcal{BS}$ with probability $C(bs_i)/\sum_{bs_j \in \mathcal{BS}} C(bs_j)$ for each request $r_l$;
14:    end if
15:    for each BS $bs_i$ with assigned requests do
16:        Observe the delay of processing request $r_l$ and update its mean value $\theta_{li}$;
17:    end for
18:    for each request $r_l$ do
19:        Observe the data traffic $\rho_l^{bst}(t)$ and update its mean value $\phi_l$;
20:    end for
21:    Observe the negative of the objective as reward and update its mean value $\psi_{t+1}(\gamma)$; update the counter $n_{t+1}(\gamma)$;
22: end for
```

5.3. Algorithm analysis

In the following we analyze the regret and time complexity of algorithms OL_UN and OL_ADA.

Theorem 2. *Given an MEC network $G = (\mathcal{BS}, E)$, a set R of requests and a set $\mathcal{S}$ of services, assuming that $C(bs_i)/\sum_{bs_j \in \mathcal{BS}} C(bs_j) \le 1 - \gamma$ and $\phi_l \cdot \delta_{unit} \le \alpha C(bs_i)$, there are online learning algorithms, i.e. algorithms* OL_UN *and* OL_ADA*, with a bounded regret of $\sigma \log \frac{T-1}{e^{1/c}+1}$ for the the dynamic service caching problem with both bursty data traffic and uncertain processing delays, while the maximum probability of violating the computing resource capacity of any BS by $(1+q)$ times is $\exp(-\frac{2q^2}{\alpha^2|R|})$, where σ is the gap between the optimal service caching and the worst service caching, γ is a threshold in the range of $(0, 1-e^{-1/c})$ with $0 < c < 1$, and α is a given constant in the range of $(0,1]$. The running time of algorithms* OL_UN *and* OL_ADA *is $O((|\mathcal{BS}|(|R| + |\mathcal{S}|))^{\omega} B)$ for a single time slot $t \in T$, where ω is the exponent of matrix multiplication and the current value of $\omega \approx 2.37$, and B is the input bits [55].*

Proof. We first show the regret bound of the proposed algorithms. The derivation of σ is similar to Theorem 1. Recall that algorithms OL_UN and OL_ADA adopts the same exploration-and-exploitation strategy as that of algorithm OL_GD. According to Theorem 1, the regret of algorithm OL_UN is

$$\sigma \log \frac{T-1}{e^{1/c}+1}. \tag{36}$$

For algorithm OL_ADA, we know that it dynamically adjusts the value of γ that has the same range as algorithm OL_UN, i.e., $[0, 1-e^{-1/c}]$, where $0 < c < 1$, the regret of algorithm OL_ADA is $\sigma \log \frac{T-1}{e^{1/c}+1}$ as well.

We now show the resource violation ratio of each BS and its probability. The proof follows similar steps in Lemma 1.

Denote by H_i' the amount of computing resource that is used to process the requests that are assigned to BS bs_i in time slot t. Clearly, we have

$$\begin{aligned} E(H_i') &= \sum_{r_l \in R} x_{li}^{\prime *} \cdot E(\rho_l^{bst}(t)) \cdot \delta_{unit} \\ &= \sum_{r_l \in R} x_{li}^{\prime *} \cdot \phi_l \cdot \delta_{unit}. \end{aligned} \tag{37}$$

Following the steps in Lemma 1, we have

$$\begin{aligned} E(H_i') &= E_1(H_i') + E_2(H_i') \\ &\le (1 - \frac{1}{t}) C(bs_i) + \frac{1}{t} C(bs_i) \\ &\le C(bs_i). \end{aligned} \tag{38}$$

The computing resource constraint of BS bs_i is violated by $(1+q)$ times with the probability

$$\begin{aligned} Pr[H_i' \ge (1+q)C(bs_i)] &\le \exp(-\frac{2q^2(C(bs_i))^2}{\sum_{bs_j \in \mathcal{BS}} \sum_{x_{li}' \ge \gamma} (\phi_l \cdot \delta_{unit} - 0)^2}) \\ &\le \exp(-\frac{2q^2}{\alpha^2 |R|}). \end{aligned} \tag{39}$$

The time complexity of OL_UN is equivalent to that of OL_GD since they share similar procedures. Algorithm OL_ADA introduces an additional procedure to determine the threshold γ, which can be divided into activation part and selection part. The activation part is an interval covering problem that can be solved in $O(\kappa \log \kappa)$ by sweep line algorithm [57], and the selection part takes $O(\kappa)$ time, where κ is the number of discretized candidates of γ. Since $(|\mathcal{BS}|(|R| + |\mathcal{S}|))^{\omega} B \gg \kappa \log \kappa$, the time complexity of OL_ADA keeps $O((|\mathcal{BS}|(|R| + |\mathcal{S}|))^{\omega} B)$ following Theorem 1.

The theorem holds.

6. GAN-guided heuristic for the dynamic service caching problem with bursty data traffic and uncertain processing delays

Precise prediction methods can maximally reduce the negative impact of decision makings under uncertainties. However, precise predictions normally need large amount of historical data on user features for accurate training, which is usually hard to obtain. Instead of focusing on obtaining enough data, we here adopt a prediction method based on small data samples of user features, i.e., GAN. Specifically, we propose a GAN-guided heuristic for the dynamic service caching problem with demand and processing delay uncertainties.

6.1. Basic idea

The basic idea of the proposed algorithm is to predict the data volumes of each request r_l in each time slot t according to the historical information. Given the predicted value, we then invoke algorithm OL_ADA to obtain a solution for request assignment and service caching. We observe that the burstiness of a request's data volume depends on many hidden features, such as its location, its registered BS, and the current social events in its location. In particular, users in the same location may have similar distributions of their data volumes. For example, a few users may be playing the same VR game, and their data volumes can be similar to each other. However, the data samples of user hidden features are usually small and hard to obtain. How to predict data volumes of users based on small samples of user data is challenging. The technique of GAN is emerging as a key enabling prediction method to deal with small samples. GAN essentially adopts the concept of a non-cooperative game in which two deep neural networks, a generator ($\mathcal{G}$) and a discriminator ($\mathcal{D}$), are trained to play against each other. The objective of $\mathcal{G}$ is to generate data volumes resembling to samples that are generated from true data distributions, while $\mathcal{D}$'s purpose is to distinguish between samples drawn from $\mathcal{G}$ and samples drawn from the true data distribution. Also, the generator uses noise z unrestrictedly within the GAN model. Our basic idea is to allow the generator to predict the data volumes of all requests in R, and $\mathcal{D}$ evaluates whether the prediction is correct or wrong. The process continues until $\mathcal{G}$ can predict the correct data volumes of requests with high probabilities. On the other hand, it is better for the prediction model to consider the variety of many hidden features of user requests. It is known that Mutual Information GAN (InfoGANs) are capable of handling such user data with various features. We thus adopt InfoGANs, whose generator and discriminator utilize Recurrent Neural Networks (RNNs). Specifically, we adopt Long Short-Term Memory (LSTM) networks [58] for both the generator and discriminator components. The choice of LSTM is motivated by its enhanced ability to capture long-term dependencies compared to traditional RNNs, which is crucial in generating and discriminating complex sequences effectively. Subsequently, learned features can be decomposed and explained by mutual information, which is denoted by I.

The novelties of this model include (1) the adoption of InfoGAN to process time series data of user information, (2) the proposal of a novel feature dimension through the prior probability distribution of latent variables. Simultaneously, latent information can control the flexibility of GAN model itself and provide an explanatory characteristic dimension, thereby preventing model collapse.

6.2. GAN model

In the Info-RNN-GAN model, we use C (latent) to represent the hidden features of users, such as the user group tag inferred from user information (location), which optimizes the learning direction of generator towards the desired data volume. In this paper, we preprocess the location of the data with one-hot encoding [59] and then treat it as the value of C, with c^t representing the coding of their locations in time slot t. It means that we can use data feature to evaluate data correctness. Another input of each cell is the noise vector z^t in each time slot t, as shown in Fig. 3.

Because of the continuous user activities, the prediction of their data volumes can be affected by historical chronological features. We thus adopt LSTM [58] to learn from these historical chronological features. Specifically, generator ($\mathcal{G}$) adopts an LSTM to learn the features of user requests. Then, softmax [60] is used to predict the data volume of the user in real time. The output is the predicted data volumes of requests in R denoted by $\mathcal{G}(z^t, c^t)$, which is also considered as *fake data*. Let $\rho_l(t)$ be the ***true data*** in the historical data samples of user requests. Discriminator $\mathcal{D}(\mathcal{G}(z^t, c^t))$ uses an LSTM to evaluate how close the fake data is from the true data. Let Q represent the features predicted by the model, with each c'^t in the vector indicating the encoding of the predicted user locations. The discriminator $\mathcal{D}$ then uses Q to determine whether it is similar to latent C. In a monitoring period T, the model continuously generates and evaluates the data of user requests. At the end of a monitoring period, the evaluation result are gathered for calculating the loss to update the model. To evaluate the closeness of the false data and true data in a monitoring period T, we use the following loss function,

$$V'(\mathcal{D}, \mathcal{G}) = \frac{1}{T}\sum_{t=1}^{T}[\log \mathcal{D}(\rho_l(t)) + \log(1 - \mathcal{D}(\mathcal{G}(z^t, c^t)))], \tag{40}$$

where $\mathcal{D}(\rho_l(t))$ represents the discriminant result of true data and T is the length of a monitoring period.

Recall that the mutual information needs to be considered, we then extend the above loss function by

$$\min_{\mathcal{G}} \max_{\mathcal{D}} V(\mathcal{D}, \mathcal{G}) = V'(\mathcal{D}, \mathcal{G}) - \frac{\lambda}{T}\sum_{t=1}^{T} I(c^t; \mathcal{G}(z^t, c^t)), \tag{41}$$

where λ is a given constant, $V'(\mathcal{D}, \mathcal{G})$ is the loss function of the discriminator $\mathcal{D}$ and generator $\mathcal{G}$ of the Info-RNN-GAN model, and $I(c^t; \mathcal{G}(z^t, c^t))$ is the reduction of uncertainty in c^t when $\mathcal{G}(z^t, c^t)$ is observed. If the value of $I(c^t; \mathcal{G}(z^t, c^t))$ is larger, they are more relevant.

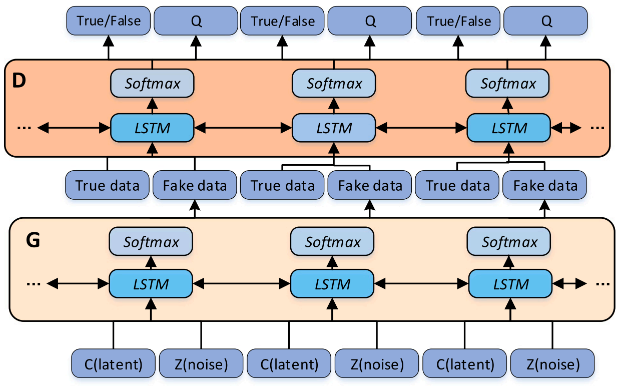


Fig. 3. Info-RNN-GAN. The generator ($\mathcal{G}$) produces sequences of predicted data volumes of requests. The discriminator ($\mathcal{D}$) is trained to distinguish between real data volume and generated data volume.

Since the distributions of the mutual information are hard to maximize directly for a disentangled representation of results, we use a lower bound to estimate the value of the mutual information by generating the direction $Q(c'^t \mid \rho_l(t))$ to approximate $P(c^t \mid \rho_l(t))$, where $Q(c'^t \mid \rho_l(t))$ represents the distribution of hidden codes in fake data and $P(c^t \mid \rho_l(t))$ represents the distribution of hidden coding in true data. Denote by $L_1(\mathcal{G}, Q)$ the lower bound of $I(c^t; \mathcal{G}(z^t, c^t))$, which can be calculated by

$$\begin{aligned} L_1(\mathcal{G}, Q) = {} & E_{\rho_l(t)\sim\mathcal{G}(z^t,c^t)}[E_{c'^t\sim P(c^t|\rho_l(t))}[\log Q(c'^t \mid \rho_l(t))]] \\ & + H(c^t) < I(c^t; \mathcal{G}(z^t, c^t)), \end{aligned} \tag{42}$$

where the entropy $H(c^t)$ is assumed to be a given constant for simplicity.

The loss function can be approximately solved as

$$\min_{\mathcal{G},Q} \max_{\mathcal{D}} V(\mathcal{D}, \mathcal{G}, Q) = V'(\mathcal{D}, \mathcal{G}) - \lambda\frac{1}{m}\sum_{t=1}^{m} L_1(\mathcal{G}, Q)). \tag{43}$$

The proposed model enables us to continuously optimize the generated value based on the training model and probability distribution of true data. The model has two advantages: (i) we use an LSTM to learn the data features of the sequence in the generator; (ii) we control the direction of generated data for hidden features based on mutual information. Therefore, we will eventually get the predicted data volumes of requests as the loss function approaches a stable status.

6.3. Algorithm description

We now describe the proposed algorithm. For each request r_l, its data volume may be related to the context or environment that can be extracted from its historical data. It is known that a RNN has the ability of exploring temporal dynamic behavior, by building connections between nodes from a directed graph along a temporal sequence. We thus adopt the Info-RNN-GAN model, which is a continuous recurrent network with adversarial training [61], where the discriminator consists of a RNN, allowing it to extract serial information from historical data for handling the uncertainties of the burstiness of user requests to make decisions. In the pre-training stage, each series representing the data volume of a request is sliced into segments of $[t, t+\zeta+1]$ through a sliding window for each time slot $t \in [1, T-\zeta-1]$, where $\zeta+1$ is the length of segments. The true data, discernible by the discriminator $\mathcal{D}$, is constituted by these partitioned sequences. Concurrently, the generator $\mathcal{G}$ predicts and fabricates series of $[t, t+\zeta+1]$, deemed as fake data, from an input integrated with segment $[t, t+\zeta]$ and the request information c^t. The discriminator $\mathcal{D}$ discriminates between true and fake data. Additionally, it endeavors to recapture the request information c^t, thereby completing the pre-training process. Then the generator $\mathcal{G}$ is used to predict the data volumes of requests based on the historical data and request information. Given the predicted values of data volume for each request r_l in time slot t, the algorithm then caches the services and assigns all requests, by invoking **Algorithm** OL_ADA. The discriminator $\mathcal{D}$ then evaluates the quality of the prediction and feeds the information to the generator $\mathcal{G}$. The generator then re-generates its predictions for the next time slot based on the feedback. The detailed steps of the proposed algorithm are shown in **Algorithm** 4, which is referred to as OL_GAN.

Algorithm 4 OL_GAN

```
Input: $G = (\mathcal{BS}, E)$, a set of services $\mathcal{S}$, and a set of requests $R$.
Output: The caching of each service $S_k \in \mathcal{S}$ and the assignment of each
    request $r_k \in R$ to its cached service in a BS.
1:  for $t \leftarrow 1, \cdots, T$ do
2:    for each request $r_l$ do
3:      if $t \leq \zeta$ then
4:        Use the mean value $\phi_l$ as the predicted data volume of request $r_l$
          for each request;
5:      else
6:        Generator $\mathcal{G}$ predicts the data volume $\rho_l^{bst}(t)$;
7:      end if
8:    end for
9:    Invoke algorithm OL_ADA to obtain the service caching and request
      assignment decision;
10:   for each BS $bs_i$ with assigned requests do
11:     Observe the processing delay and update its mean value $\theta_{li}$;
12:   end for
13:   for each request $r_l$ do
14:     Observe the data traffic $\rho_l^{bst}(t)$ and update its mean value $\phi_l$;
15:     Discriminator $\mathcal{D}$ observes the real data volume of $r_l$ and calculates
        its loss;
16:   end for
17: end for
```

We then show the time complexity of algorithm OL_GAN in the following theorem.

Theorem 3. *The time complexity of algorithm* OL_GAN *is* $O(\zeta|R|(|z^t| + |c^t|)^2 + (|\mathcal{BS}|(|R| + |\mathcal{S}|))^{\omega}B)$ *for a single time slot* $t \in T$, *where* ω *is the exponent of matrix multiplication and the current value of* $\omega \approx 2.37$, *and* B *is the input bits [55].*

Proof. The assignment in a single time slot t primarily involves two time-consuming parts: First, the inference that the generator $\mathcal{G}$ predicts the bursty data traffic of user requests, and second, the invocation of algorithm OL_ADA. Since the generator G adopts an LSTM to learn the features of user requests, the inference time complexity of LSTM is $O(\zeta d^2)$ [58], where ζ is the length of the input time series data, $d = |z^t|+|c^t|$ is the width of the input vector for a single time slot. Therefore, the overall inference time complexity of the generator $\mathcal{G}$ is $O(\zeta|R|(|z^t|+|c^t|)^2)$ for all user requests in a single time slot. Following Theorem 2, the time complexity of OL_ADA is $O((|\mathcal{BS}|(|R| + |\mathcal{S}|))^{\omega}B)$. So the time complexity of OL_GAN thus is $O(\zeta|R|(|z^t|+|c^t|)^2+(|\mathcal{BS}|(|R|+|\mathcal{S}|))^{\omega}B)$.

7. Experiments

We evaluate the proposed algorithms by experimental simulations in both synthetic and real networks.

7.1. Parameter settings

We consider an MEC network G consisting of from 10 to 100 BSs, where each network topology is generated by GT-ITM [62]. Each BS has a computing capacity in the range of from four to eight CPU cores working at a frequency of 2 GHz, and the average delay in a BS is a value that is randomly drawn in the range of $[10, 20]$ milliseconds [63, 64]. The request data is sampled from the data trace of Universidad Del Cauca network flow collections [65]. We filter out network flow with less than 100 records, and map the service type of the remaining flows to six kinds of randomly generated service type with service instantiation time randomly drawn in the range of [0.5, 1] seconds. For pre-training the model of algorithm OL_GAN, the segment length is set to $\zeta = 10$, and the monitoring period T is set to 50 for balancing the training speed and stability. Unless otherwise specified, we will adopt these default settings in our experiments. To smooth the results, each line in the figure is averaged by 50 runs with different random seeds.

Benchmarks: We compare the proposed algorithms with the following three algorithms:

- Greedy_GD: A greedy approach for which each BS greedily selects a service and its tasks that could minimize the delay of each request, assuming that the data traffic of each request is given.
- Pri_GD: A priority-driven service caching algorithm in [2] that assigns each request a priority according to the number of BSs covered in its transmission range, and the BS takes priority in processing the high priority request.
- OL_Reg: An online algorithm with a single autoregression prediction that predicts the bursty data traffic following an autoregressive moving average (ARMA) model. OL_UN is then invoked for service caching and request assignment. Let $\hat{\rho}_l(t)$ be the predicted bursty data traffic of request r_l in time slot t. Using the information of the previous p time slots, assuming that the value of p is given, $\hat{\rho}_l(t)$ can be predicted by the following ARMA model,

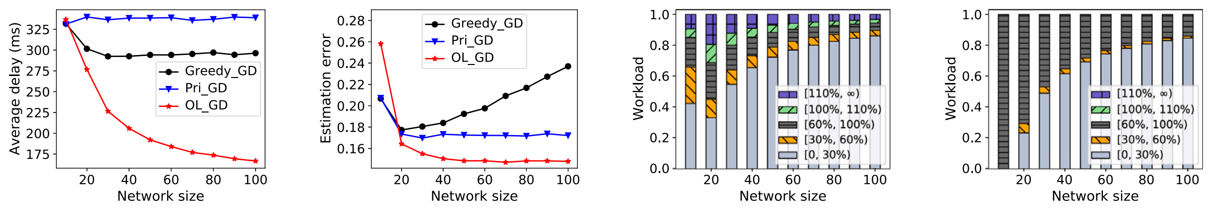
$$\hat{\rho}_l(t) = a_1 \cdot \rho_l(t-1) + a_2 \cdot \rho_l(t-2) + \cdots + a_p \cdot \rho_l(t-p), \tag{44}$$

where $a_{p'}$ is a constant with $0 \leq a_{p'} \leq 1$, $\sum_{l=1}^{p} a_l = 1$, and $a_{p_1} \geq a_{p_2}$ if $p_1 < p_2$.

7.2. Performance of algorithm OL_GD

We first evaluate the performance of algorithm OL_GD against algorithms Pri_GD, and Greedy_GD in terms of average delays, estimation errors, and workloads, with the number of BSs from 10 to 100 of a finite time horizon of 50 time slots. The estimation error measures how much the delays estimated by the algorithm based on its assignments deviate from their actual delays. Fig. 4(a) shows the average delays obtained by algorithms OL_GD, Greedy_GD, and Pri_GD. It shows that the average delays of OL_GD and Greedy_GD decrease as the number of BSs grows. The reason is that the computing resources in BSs are increasing with the growth of BS number. As the result, more services can be cached into the network, thereby increasing the probability of implementing requests in BSs with shorter execution delay. Moreover, OL_GD obtains similar average delay as Greedy_GD and Pri_GD in networks with 10 BSs, and gets 44% lower delay than Greedy_GD in networks with 100 BSs. It can be explained by comparing Figs. 4(c) and 4(d). From Fig. 4(d), we can see that in networks with 10 BSs, all BSs are serving requests at full workloads, meaning the assumption that the overall resources of all BSs is higher than the total resource demand of all requests is violated. In this case, the random exploration process in OL_GD cannot provide good performance until the total resource of BSs grow with the size of the network. Furthermore, the average estimation error shown in Fig. 4(b) illustrates the efficiency of the exploration process in OL_GD, especially when the number of BSs is growing.

We then investigate the performance of OL_GD, Pri_GD, and Greedy_GD by varying the number of user requests arriving in the system from 20 to 160, on a fixed network topology named ITM-50. As shown in Fig. 5, ITM-50 is a network topology generated by GT-ITM [62] with 50 BSs. The results are shown in Fig. 6. Fig. 6(a)



(a) Average Delay. (b) Estimation Error. (c) BS workload of `OL_GD`. (d) BS workload of `Greedy_GD`.

Fig. 4. The performance of algorithms `OL_GD`, `Greedy_GD`, `Pri_GD` in networks with the number of BSs being varied from 10 to 100.

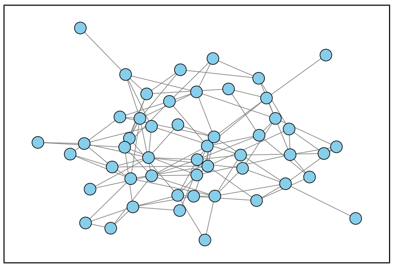


Fig. 5. Network topology ITM-50 with 50 BSs, generated by GT-ITM.

shows that the average delays obtained by `OL_GD` and `Greedy_GD` increase as the number of user requests grows, and `OL_GD` increases faster than `Greedy_GD`. The rationale behind is that the more user requests require more computing resources for processing request data. Although `OL_GD` is better at assigning requests to low-latency BSs than `Greedy_GD`, it becomes more difficult to find suitable low-latency BSs as the number of requests increases. Fig. 6(b) illustrates that the estimation errors decrease as the number of requests grows. It can be explained that a higher number of requests enables the collection of more historical information about BSs, which helps to estimate the real processing speed of BSs more accurately.

We further evaluate convergence process of the algorithms `OL_GD`, `Greedy_GD`, and `Pri_GD` in a finite time horizon of 50 time slots with 50 BSs. The results are illustrated in Fig. 7. Fig. 7(a) shows that the average delays of `OL_GD` and `Greedy_GD` are decreasing as time goes, and `OL_GD` converges rapidly. This is because the proposed algorithm `OL_GD` adopts the MAB technique that exploring the distribution of processing delay more efficiently. Instead, algorithms `Greedy_GD` and `Pri_GD` cache services and offload user tasks without an intentionally designed exploration process that cannot trade off between exploration and exploitation to take the full utilization of the historical information of processing delays. Furthermore, The average estimation error shown in Fig. 7(b) illustrates that the benchmarks `Greedy_GD` and `Pri_GD` achieve the average estimation error gap no less than 22% compared with `OL_GD`, due to the inefficient exploration process.

The average estimation error shown in Fig. 7(b) illustrates that the benchmarks `Greedy_GD` and `Pri_GD` achieve the gap of average estimation error no less than 20% compared with `OL_GD`, due to the inefficient exploration process.

We then study the performance of the proposed algorithms on a real network GÉANT [66] in a time horizon of 50 time slots. The results are shown in Fig. 8. From Fig. 8, we can see that algorithm `OL_GD` achieves lower network delay and estimation error than algorithms `Greedy_GD` and `Pri_GD` in real network GÉANT. However, the running time of `OL_GD` is much higher than that of `Greedy_GD` and `Pri_GD`, but is still acceptable to provide real-time assignments.

7.3. Performance of algorithms `OL_UN`, `OL_ADA`, and `OL_GAN`

We evaluate the performance of algorithms `OL_UN`, `OL_ADA`, `OL_GAN`, and `OL_Reg`, assuming that the data traffic of user requests is not given, by varying the network size from 10 to 100. Note that algorithms `OL_REG` and `OL_GAN` require a minimum number of historical samples to feed the model. In the first 10 time slots, we assign requests based on the mean values of historical data (i.e., ϕ_l for each request r_l) rather than the predicted values. Fig. 9(a) shows the average delay, from which we can see that algorithm `OL_ADA` obtains similar average delay with `OL_Reg`. This is due to the fact that `OL_Reg` needs a lot of data to make accurate predictions, which counterbalances the potential benefits of the predictions, particularly in the context of larger networks. It is further supported by the estimation errors shown in Fig. 9(b). Although algorithm `OL_ADA` adopts the zooming algorithm to optimize filtering threshold γ, it cannot accurately predict the data traffic of user requests, thus occurs estimation error and leads to sub-optimal solutions. Besides, `OL_GAN` has a lower average delay than algorithm `OL_Reg`. The rationale behind is that algorithm `OL_Reg` adopts a simple ARMA model to predict user data traffic, and the prediction method replies on the historical data. Usually, the accuracy of ARMA model will be higher if there is more historical information. In contrast, algorithm `OL_GAN` adopts a GAN-based method that works very well in small volume of historical data.

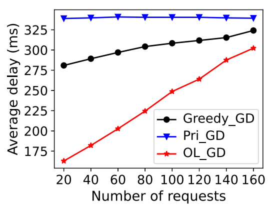
The running times of both algorithms are shown in Fig. 9(c), from which we can see that algorithm `OL_GAN` has around 50% lower average running time than that of algorithm `OL_Reg`. The rationale is that `OL_GAN` is a pre-trained model that the inference time is short and fixed, while `OL_Reg` cannot be pre-trained and the automatic parameter tuning for ARMA model is time-consuming.

We further evaluate the performance of `OL_UN`, `OL_ADA`, `OL_GAN`, and `OL_Reg` by varying the number of user requests arriving in the system from 20 to 160, on a fixed network topology ITM-50. The results are shown in Fig. 10. Fig. 10(a) shows that `OL_ADA` and `OL_Reg` obtain similar average delays, and `OL_GAN` obtains the lowest average delays. The reason is that `OL_Reg` predicts the data volumes of user requests by adopting the ARMA model, but does not dynamically adjust the filtering threshold γ like `OL_ADA`. `OL_GAN` achieves lower average delays by predicting more accurately than `OL_Reg`. Fig. 10(b) illustrates that `OL_Reg`, `OL_ADA`, and `OL_GAN` obtain lower estimation errors than `OL_UN`. This is because `OL_ADA` dynamically adjusts the filtering threshold γ for accelerating the exploration of BSs, while `OL_Reg` and `OL_GAN` predict the data volumes of user requests for accurate estimation.

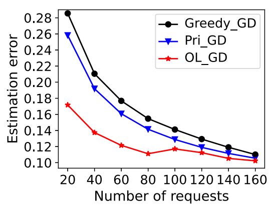
We then investigate the performance of the algorithms `OL_UN`, `OL_ADA`, `OL_GAN`, and `OL_Reg` in a network of 50 BSs of a finite time horizon of 50 time slots. The results are depicted in Fig. 11, from which we can see that algorithms `OL_Reg` and `OL_GAN` get a cliff descent on average delays in time slot 11. This is because algorithms `OL_Reg` and `OL_GAN` predict the data volumes of requests via ARMA and GAN, respectively, instead of directly using the mean values of historical data. Furthermore, the heightened precision in the predictions delivered by `OL_GAN` results in a lower average delay compared to that of `OL_REG`.

8. Conclusion and future works

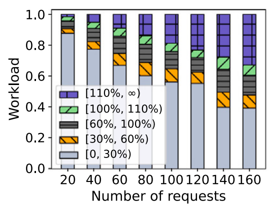
In this paper we studied the problem of dynamic service caching and task offloading in MEC networks, by considering the bursty data



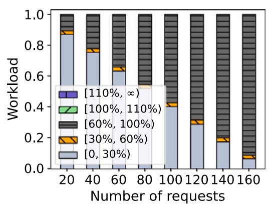
(a) Average Delay.



(b) Estimation Error.

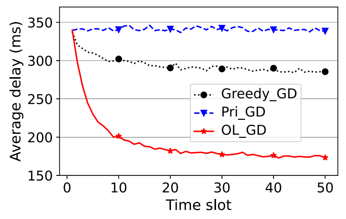


(c) BS workload of OL_GD.

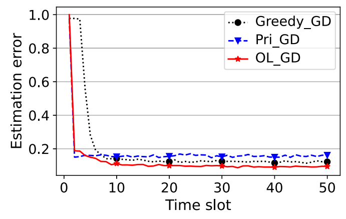


(d) BS workload of Greedy_GD.

Fig. 6. The performance of algorithms OL_GD, Greedy_GD, Pri_GD in network ITM-50 with the number of user requests being varied from 20 to 160.



(a) Average Delay.



(b) Average Estimation Error.

Fig. 7. The performance of algorithms OL_GD, Greedy_GD, Pri_GD in a period of 50 time slots with 50 BSs.

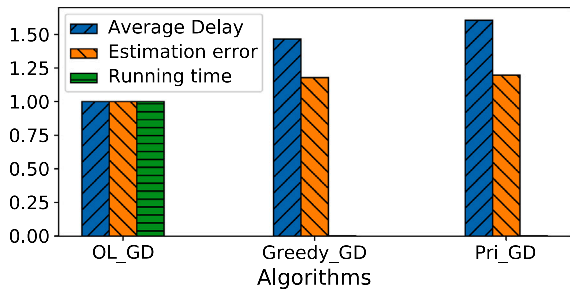
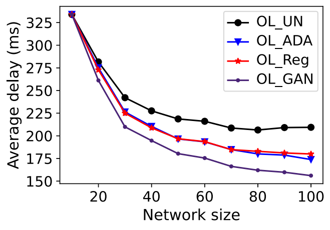
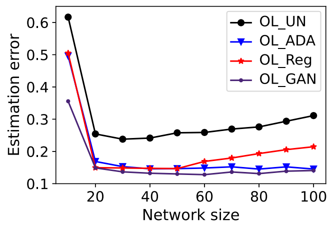


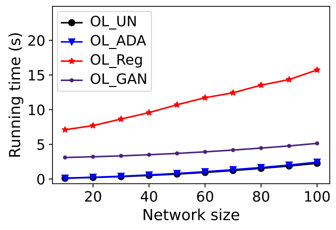
Fig. 8. The performance improvement ratio of algorithms OL_GD, Greedy_GD, Pri_GD against Greedy_GD on a real topology GÉANT.



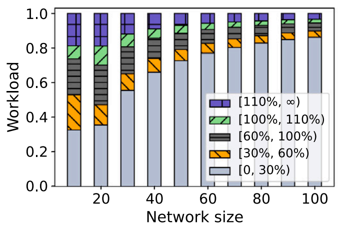
(a) Average delay.



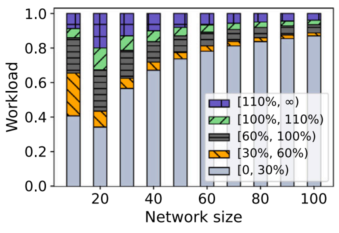
(b) Estimation error.



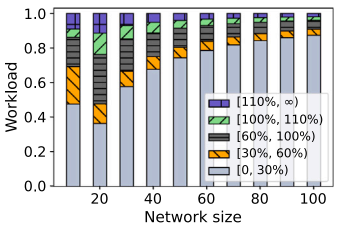
(c) Running time.



(d) BS workload of OL_ADA.



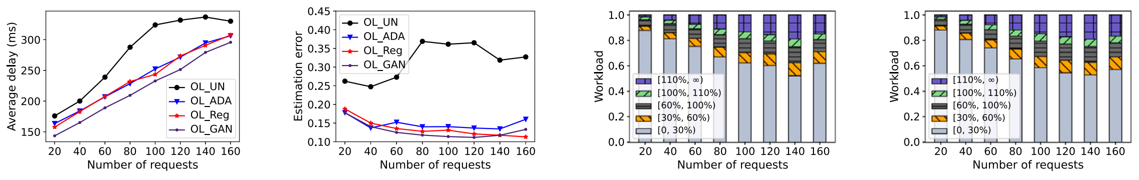
(e) BS workload of OL_Reg.



(f) BS workload of OL_GAN.

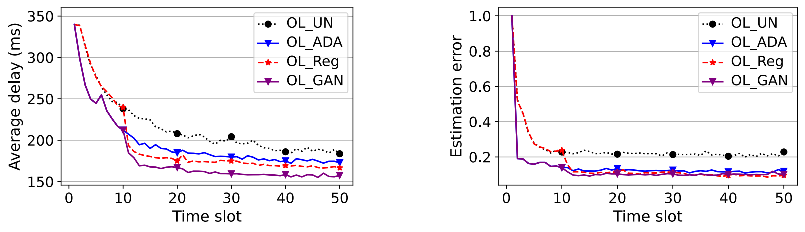
Fig. 9. The performance of algorithms OL_UN, OL_ADA, and OL_Reg in networks with the number of BSs being varied from 10 to 100.

traffic and uncertain processing delays. For the problem with uncertain processing delays, we first proposed an online learning algorithm based on the MAB method, and analyzed the regret bound of the algorithm too. When both user demands and processing delays are uncertain, For the problem with both bursty data traffic and uncertain processing delays, we then devised an online learning algorithm with a bounded regret, via leveraging the zooming technique. We also proposed a novel prediction method to accurately predict the user demands based on the state-of-the-art GAN model. Based on the proposed prediction model, we devised an efficient heuristic for the problem with bursty data traffic and uncertain processing delays. We finally investigated the performance of the proposed algorithms by simulations, using a



(a) Average delay. (b) Estimation error. (c) BS workload of `OL_ADA`. (d) BS workload of `OL_GAN`.

Fig. 10. The performance of algorithms `OL_UN`, `OL_ADA`, and `OL_Reg` in network ITM-50 with the number of user requests being varied from 20 to 160.



(a) Average Delay. (b) Estimation error.

Fig. 11. The performance of algorithms `OL_UN`, `OL_ADA`, and `OL_Reg` in a period of 50 time slots.

real data trace. Experiment results show that the performance of the proposed algorithms outperforms the comparison algorithms by up to 44% in terms of average delay.

The future works of this paper include the following directions. First, we assumed that the basic data traffic within a finite time horizon is given as a priori. The length of such a finite time horizon determines the values of such basic data traffic and may rely on different types of applications. In our future work, we thus will explore the impacts of the lengths of time horizon on the basic data traffic of different types of requests. Second, the data traffic of requests may have varying changing patterns in different time scales. We thus will also extend the proposed GAN model to predict the data traffic of requests that have different temporal patterns within different time scales.

Funding

The researches of Zichuan Xu and Qiufen Xia are funded by National Nature Science Foundation of China under Grants 62172071, 62172068, in part by Natural Science Foundation of Shandong Province under Grants No.ZR2023LZH008, No.ZR2023LZH013, in part by Natural Science Foundation of Liaoning Province under Grant 2023-MS-111.

CRediT authorship contribution statement

Wenhao Ren: Conceptualization, Software, Validation, Writing – original draft. **Zichuan Xu:** Conceptualization, Methodology, Writing – review & editing. **Weifa Liang:** Formal analysis, Writing – review & editing. **Haipeng Dai:** Conceptualization, Methodology, Writing – review & editing. **Omer F. Rana:** Visualization, Writing – review & editing. **Pan Zhou:** Investigation, Writing – review & editing. **Qiufen Xia:** Formal analysis, Funding acquisition, Writing – review & editing. **Haozhe Ren:** Software. **Mingchu Li:** Writing – review & editing. **Guowei Wu:** Writing – review & editing.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Zichuan Xu, Qiufen Xia reports financial support was provided by National Natural Science Foundation of China, Natural Science Foundation of Shandong Province, Natural Science Foundation of Liaoning Province. Other authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

[1] Y. Liang, J. Ge, S. Zhang, J. Wu, Z. Tang, B. Luo, A utility-based optimization framework for edge service entity caching, IEEE Trans. Parallel Distrib. Syst. 30 (11) (2019) 2384–2395.

[2] Q. Xie, Q. Wang, N. Yu, H. Huang, X. Jia, Dynamic service caching in mobile edge networks, in: Proc. of MASS, IEEE, 2018.

[3] N. Bronson, et al., TAO: Facebook's distributed data store for the social graph, in: Proc. of ATC, USENIX, 2013.

[4] P. Jin, J. Guo, Y. Xiao, R. Shi, Y. Niu, F. Liu, C. Qian, Y. Wang, PostMan: Rapidly mitigating bursty traffic by offloading packet processing, in: Proc. of ATC, USENIX, 2019.

[5] W. Miao, G. Min, Y. Wu, H. Wang, J. Hu, Performance modelling and analysis of software-defined networking under bursty multimedia traffic, ACM Trans. Multimed. Comput. Commun. Appl. 12 (5s) (2016) 1–19.

[6] X. Zhou, Z. Zhao, R. Li, Y. Zhou, T. Chen, Z. Niu, H. Zhang, Toward 5G: when explosive bursts meet soft cloud, IEEE Netw. 28 (6) (2019) 12–17.

[7] N. Eshraghi, B. Liang, Joint offloading decision and resource allocation with uncertain task computing requirement, in: Proc. of INFOCOM, IEEE, 2019.

[8] M.V. Barbera, S. Kosta, A. Mei, J. Stefa, To offload or not to offload? The bandwidth and energy costs of mobile cloud computing, in: Proc. of INFOCOM, IEEE, 2013.

[9] M. Chen, Y. Qian, Y. Hao, Y. Li, J. Song, Data-driven computing and caching in 5G networks: Architecture and delay analysis, IEEE Wirel. Commun. 25 (1) (2018) 70–75.

[10] M. Chen, Y. Hao, Y. Li, C.F. Lai, D. Wu, On the computation offloading at ad hoc cloudlet: Architecture and service modes, IEEE Commun. Mag. 53 (6) (2015) 18–24.

[11] J. Xu, L. Chen, P. Zhou, Joint service caching and task offloading for mobile edge computing in dense networks, in: Proc. of INFOCOM, IEEE, 2018.

[12] W. Chu, X. Jia, Z. Yu, J.C.S. Lui, Y. Lin, Joint service caching, resource allocation and task offloading for MEC-based networks: A multi-layer optimization approach, IEEE Trans. Mob. Comput. 23 (4) (2024) 2958–2975.

[13] N. Chen, S. Zhang, J. Wu, Z. Qian, S. Lu, Learning scheduling bursty requests in mobile edge computing using DeepLoad, Comput. Netw. 184 (2021).

[14] M. Liu, F.R. Yu, Y. Teng, V.C.M. Leung, M. Song, Joint computation offloading and content caching for wireless blockchain networks, in: Proc. of INFOCOM, IEEE, 2018.

[15] H. Wang, R. Li, L. Fan, H. Zhang, Joint computation offloading and data caching with delay optimization in mobile-edge computing systems, in: Proc. of WCSP, IEEE, 2017.

[16] T.X. Tran, K. Chan, D. Pompili, COSTA: Cost-aware service caching and task offloading assignment in mobile-edge computing, in: Proc. of SECON, IEEE, 2019.

[17] Z. Xu, L. Zhou, S. Chau, W. Liang, Q. Xia, P. Zhou, Collaborate or separate? Distributed service caching in mobile edge clouds, in: Proc. of INFOCOM, IEEE, 2020.

[18] N. Zhang, S. Guo, Y. Dong, D. Liu, Joint task offloading and data caching in mobile edge computing networks, Comput. Netw. 182 (2020).

[19] R. Fan, B. Liang, S. Zuo, H. Hu, H. Jiang, N. Zhang, Robust task offloading and resource allocation in mobile edge computing with uncertain distribution of computation burden, IEEE Trans. Commun. 71 (7) (2023) 4283–4299.

[20] X. Wang, J. Ye, J.C.S. Lui, Decentralized task offloading in edge computing: A multi-user multi-armed bandit approach, in: Proc. of INFOCOM, IEEE, 2022.

[21] Z. Xu, W. Liang, M. Jia, M. Huang, G. Mao, Task offloading with network function services in a mobile edge-cloud network, IEEE Trans. Mob. Comput. 18 (11) (2019) 2672–2685.

[22] Z. Xu, S. Wang, S. Liu, H. Dai, Q. Xia, W. Liang, G. Wu, Learning for exception: Dynamic service caching in 5G-enabled MECs with bursty user demands, in: Proc. of ICDCS, IEEE, 2020.

[23] L. Xiao, X. Wan, C. Dai, X. Du, X. Chen, M. Guizani, Security in mobile edge caching with reinforcement learning, IEEE Wirel. Commun. 25 (3) (2018) 116–122.

[24] X. Chen, L. Jiao, W. Li, X. Fu, Efficient multi-user computation offloading for mobile-edge cloud computing, IEEE/ACM Trans. Netw. 24 (5) (2015) 2795–2808.

[25] M. Chen, Y. Hao, L. Hu, K. Huang, V.K. Lau, Green and mobility-aware caching in 5G networks, IEEE Trans. Wireless Commun. 16 (12) (2017) 8347–8361.

[26] M. Huang, W. Liang, X. Shen, Y. Ma, H. Kan, Reliability-aware virtualized network function services provisioning in mobile edge computing, IEEE Trans. Mob. Comput. 19 (11) (2019) 2699–2713.

[27] Q. Li, S. Wang, A. Zhou, X. Ma, A.X. Liu, QoS driven task offloading with statistical guarantee in mobile edge computing, IEEE Trans. Mob. Comput. 21 (1) (2020) 278–290.

[28] M. Jia, J. Cao, L. Yang, Heuristic offloading of concurrent tasks for computation-intensive applications in mobile cloud computing, in: Proc. of INFOCOM, IEEE, 2014.

[29] H. Badri, T. Bahreini, D. Grosu, K. Yang, Energy-aware application placement in mobile edge computing: a stochastic optimization approach, IEEE Trans. Parallel Distrib. Syst. 31 (4) (2020) 909–922.

[30] Q. Xia, W. Ren, Z. Xu, X. Wang, W. Liang, When edge caching meets a budget: Near optimal service delivery in multi-tiered edge clouds, IEEE Trans. Serv. Comput. 15 (6) (2021) 3634–3648.

[31] L. Xiao, X. Lu, T. Xu, X. Wan, W. Ji, Y. Zhang, Reinforcement learning-based mobile offloading for edge computing against jamming and interference, IEEE Trans. Commun. 68 (10) (2020) 6114–6126.

[32] M. Hu, L. Zhuang, D. Wu, Y. Zhou, X. Chen, Liang. Xiao, Learning driven computation offloading for asymmetrically informed edge computing, IEEE Trans. Parallel Distrib. Syst. 30 (8) (2019) 1802–1815.

[33] Z. Cheng, M. Liwang, N. Chen, L. Huang, X. Du, M. Guizani, Deep reinforcement learning-based joint task and energy offloading in UAV-aided 6G intelligent edge networks, Comput. Commun. 192 (2022).

[34] Y. Wang, X. Han, S. Jin, MAP based modeling method and performance study of a task offloading scheme with time-correlated traffic and VM repair in MEC systems, Wirel. Netw. 29 (1) (2023) 47–68.

[35] J. Yan, S. Bi, L. Duan, Y.J.A. Zhang, Pricing-driven service caching and task offloading in mobile edge computing, IEEE Trans. Wireless Commun. 20 (7) (2021) 4495–4512.

[36] G. Zhao, H. Xu, Y. Zhao, C. Qiao, L. Huang, Offloading dependent tasks in mobile edge computing with service caching, in: Proc. of INFOCOM, IEEE, 2020.

[37] A. Lakhan, M. Ahmad, M. Bilal, A. Jolfaei, R.M. Mehmood, Mobility aware blockchain enabled offloading and scheduling in vehicular fog cloud computing, IEEE Trans. Intell. Transp. Syst. 22 (7) (2021) 4212–4223.

[38] M. Deng, H. Tian, X. Lyu, Adaptive sequential offloading game for multi-cell mobile edge computing, in: Proc. of ICT, IEEE, 2016.

[39] I. Ketykó, L. Kecskés, C. Nemes, L. Farkas, Multi-user computation offloading as multiple knapsack problem for 5G mobile edge computing, in: Proc. of EuCNC, IEEE, 2016.

[40] L. Yang, H. Zhang, M. Li, J. Guo, H. Ji, Mobile edge computing empowered energy efficient task offloading in 5G, IEEE Trans. Veh. Technol. 67 (7) (2018) 6398–6409.

[41] X. Li, X. Zhang, T. Huang, Asynchronous online service placement and task offloading for mobile edge computing, in: Proc. of SECON, IEEE, 2021.

[42] P.A. Apostolopoulos, E.E. Tsiropoulou, S. Papavassiliou, Risk-aware data offloading in multi-server multi-access edge computing environment, IEEE/ACM Trans. Netw. 28 (3) (2020) 1405–1418.

[43] M. Hu, L. Zhuang, D. Wu, Y. Zhou, X. Chen, L. Xiao, Learning driven computation offloading for asymmetrically informed edge computing, IEEE Trans. Parallel Distrib. Syst. 30 (8) (2019) 1802–1815.

[44] Z. Chen, J. Hu, G. Min, A.Y. Zomaya, T. El-Ghazawi, Towards accurate prediction for high-dimensional and highly-variable cloud workloads with deep learning, IEEE Trans. Parallel Distrib. Syst. 31 (4) (2019) 923–934.

[45] Y. Niu, P. Jin, J. Guo, Y. Xiao, R. Shi, F. Liu, C. Qian, Y. Wang, PostMan: Rapidly mitigating bursty traffic via on-demand offloading of packet processing, IEEE Trans. Parallel Distrib. Syst. 33 (2) (2021) 374–387.

[46] S. Zhao, X. Zhang, P. Cao, X. Wang, Design of robust and efficient edge server placement and server scheduling policies, in: Proc. of IWQOS, IEEE, 2021.

[47] S. Deng, C. Zhang, C. Li, J. Yin, S. Dustdar, A.Y. Zomaya, Burst load evacuation based on dispatching and scheduling in distributed edge networks, IEEE Trans. Parallel Distrib. Syst. 32 (8) (2021) 1918–1932.

[48] Z. Xu, S. Wang, S. Liu, H. Dai, Q. Xia, W. Liang, G. Wu, Learning for exception: Dynamic service caching in 5G-enabled MECs with bursty user demands, in: Proc. of ICDCS, IEEE, 2020.

[49] Intel neural compute stick, 2020, https://software.intel.com/en-us/movidius-ncs. (Accessed 2020).

[50] S. Cao, D. Liu, C. Dai, C. Wang, Y. Yang, W. Zhang, D. Zheng, Reinforcement learning based tasks offloading in vehicular edge computing networks, Comput. Netw. 234 (2023).

[51] J. Tang, T.Q.S. Quek, T.H. Chang, B. Shim, Systematic resource allocation in cloud RAN with caching as a service under two timescales, IEEE Trans. Commun. 67 (11) (2019) 7755–7770.

[52] K. Wang, W. Chen, J. Li, Y. Yang, L. Hanzo, Joint task offloading and caching for massive MIMO-aided multi-tier computing networks, IEEE Trans. Commun. 70 (3) (2022) 1820–1833.

[53] TensorFlow lite, 2020, https://www.tensorflow.org/lite/. (Accessed in Dec. 2020).

[54] M. Mitzenmacher, E. Upfal, Probability and Computing: Randomized Algorithms and Probabilistic Analysis, Cambridge University Press, 2005.

[55] M.B. Cohen, Y.T. Lee, Z. Song, Solving linear programs in the current matrix multiplication time, in: Proc. of STOC, ACM, 2019.

[56] A. Slivkins, Introduction to multi-armed bandits, 2019, [Online]. Available: http://arxiv.org/abs/1904.07272.

[57] Bentley, Ottmann, Algorithms for reporting and counting geometric intersections, IEEE Trans. Comput. 100 (6) (1979) 643–647.

[58] S. Hochreiter, J. Schmidhuber, Long short-term memory, J. Neural Comput. 9 (8) (1997) 1735–1780, MIT Press.

[59] P. Rodrìguez, M.A. Bautista, J. Gonzàlez, S. Escalera, Beyond one-hot encoding: lower dimensional target embedding, J. Image Vis. Comput. 75 (2018) 21–31.

[60] A. Martins, R. Astudillo, From softmax to sparsemax: A sparse model of attention and multi-label classification, in: Proc. of ICML, JMLR, 2016.

[61] M. Mathieu, J. Zhao, P. Sprechmann, A. Ramesh, Y. LeCun, Disentangling factors of variation in deep representation using adversarial training, in: Proc. of NIPS, 2016.

[62] GT-ITM, 2021, http://www.cc.gatech.edu/projects/gtitm/. (Accessed in June 2021).

[63] Q. Kuang, J. Gong, X. Chen, X. Ma, Age-of-information for computation-intensive messages in mobile edge computing, in: Proc. of WCSP, IEEE, 2019.

[64] R. Yu, G. Xue, X. Zhang, Provisioning qos-aware and robust applications in internet of things: a network perspective, IEEE/ACM Trans. Netw. 27 (5) (2019) 1931–1944.

[65] Universidad del cauca IP network traffic flows labeled with 75 apps, 2018, https://www.kaggle.com/jsrojas/ip-network-traffic-flows-labeled-with-87-apps. (Accessed in September 2018).

[66] Géant, 2020, http://www.geant.net/. (Accessed in February 2020).

Wenhao Ren received his B.Sc. degree from Harbin Engineering University in China in 2019 in computer science and technology. He is currently pursuing his Ph.D. degree at the Dalian University of Technology. His research interests include mobile cloud computing, big data processing, and network function virtualization.

Zichuan Xu received the B.Sc. and M.E. degrees in computer science from the Dalian University of Technology, China, in 2008 and 2011, respectively, and the Ph.D. degree from The Australian National University in 2016. From 2016 to 2017, he was a Research Associate with the Department of Electronic and Electrical Engineering, University College London, UK. He is currently a Full Professor with the School of Software, Dalian University of Technology. He is also a "Xinghai Scholar" with the Dalian University of Technology. His research interests include mobile edge computing, serverless computing, network function virtualization, and algorithm design.

Weifa Liang received the Ph.D. degree from the Australian National University in 1998, the ME degree from the University of Science and Technology of China in 1989, and the B.Sc. degree from Wuhan University, China in 1984, all in Computer Science. He is a Full Professor in the Department of Computer Science at City University of Hong Kong. Prior to that, he was a Full Professor in the Australian National University. His research interests include design and analysis of energy efficient routing protocols for wireless ad hoc and sensor networks, Mobile Edge Computing (MEC), Network Function Virtualization (NFV), Internet of Things and digital twins, design and analysis of parallel and distributed algorithms, approximation algorithms, and graph theory. He currently serves as an Editor of IEEE Transactions on Communications, and he is a senior member of IEEE.

Haipeng Dai received the B.S. degree in the Department of Electronic Engineering from Shanghai Jiao Tong University, Shanghai, China, in 2010, and the Ph.D. degree in the Department of Computer Science and Technology in Nanjing University, Nanjing, China, in 2014. His research interests are mainly in the areas of data mining, Internet of Things, and mobile computing. He is an associate professor in the Department of Computer Science and Technology in Nanjing University. His research papers have been published in many prestigious conferences and journals such as ACM VLDB, IEEE ICDE, ACM SIGMETRICS, ACM MobiSys, ACM MobiHoc, ACM UbiComp, IEEE INFOCOM, IEEE ICDCS, IEEE ICNP, IEEE SECON, IEEE IPSN, IEEE JSAC, IEEE/ACM TON, IEEE TMC, IEEE TPDS, IEEE TDSC, and IEEE TOSN. He is an IEEE and ACM member. He serves/ed as Poster Chair of the IEEE ICNP'14, Track Chair of the ICCCN'19 and the ICPADS'21, TPC member of the ACM MobiHoc'20-21, IEEE INFOCOM'20-22, IEEE ICDCS'20-21, IEEE ICNP'14, IEEE IWQoS'19-21, IEEE IPDPS'20'22 and IEEE MASS'18-19. He received Best Paper Award from IEEE ICNP'15, Best Paper Award Runner-up from IEEE SECON'18, and Best Paper Award Candidate from IEEE INFOCOM'17.

Omer F. Rana received the B.S. degree in information systems engineering from the Imperial College of Science, Technology and Medicine, London, U.K., the M.S. degree in microelectronics systems design from the University of Southampton, Southampton, U.K., and the Ph.D. degree in neural computing and parallel architectures from the Imperial College of Science, Technology and Medicine. He is a Professor of performance engineering with Cardiff University, Cardiff, U.K. His current research interests include problem solving environments for computational science and commercial computing, data analysis and management for large-scale computing, and scalability in high performance agent systems.

Pan Zhou received the B.S. degree in the advanced class of HUST, the M.S. degree from the Department of Electronics and Information Engineering, HUST, Wuhan, China, in 2006 and 2008, respectively, and the Ph.D. degree from the School of Electrical and Computer Engineering, the Georgia Institute of Technology (Georgia Tech) in 2011, Atlanta, USA. He is currently a full professor and Ph.D. advisor with Hubei Engineering Research Center on Big Data Security, School of Cyber Science and Engineering, Huazhong University of Science and Technology (HUST), Wuhan, P.R. China. He is currently an associate editor of IEEE Transactions on Network Science and Engineering. His current research interest includes: security and privacy, big data analytics, machine learning, and information networks.

Qiufen Xia received the B.Sc. and M.E. degrees in computer science from the Dalian University of Technology, China, in 2009 and 2012, respectively, and the Ph.D. degree in computer science from the Australian National University, in 2017. She is currently an Associate Professor with the Dalian University of Technology. Her research interests include mobile edge computing, query evaluation, big data analytics, big data management in distributed clouds, and network function virtualization.

Haozhe Ren received the Ph.D. degree from the Dalian University of Technology in 2023, the M.E. degree from the Xinjiang Normal University in China in 2018, and the B.Sc. degree from the University of Science and Technology Beijing in China in 2012. He is currently a postdoctoral fellow with the Dalian University of Technology. His current research interests include network function virtualization, software-defined networking, algorithmic game theory, digital twin, and optimization problems.

Mingchu Li received the B.S. degree in mathematics from Jiangxi Normal University, Nanchang, China, in 1983, the M.S. degree in applied science from the University of Science and Technology Beijing, Beijing, China, in 1989, and the Ph.D. degree in mathematics from the University of Toronto, Toronto, ON, Canada, in 1998. He was an Associate Professor with the University of Science and Technology Beijing from 1989 to 1994. From 2002 to 2004, he was a Full Professor with the School of Software, Tianjin University, Tianjin, China. Since 2004, he has been a Full Professor of the School of Software Technology, Dalian University of Technology, Dalian, China. His main research interests include mobile edge computing, theoretical computer science and information security, trust models and cooperative game theory.

Guowei Wu received his Ph.D degree from Harbin Engineering University in 2003, China. He is now a professor at the School of Software, Dalian University of Technology in China. His research interests include software-defined networking, network function virtualization, edge computing, embedded real-time systems, and cyber–physical systems. He has published over 100 papers in journals and conferences.