Profit Driven Service Provisioning in Edge Computing via Deep Reinforcement Learning

Yuchen Li, Weifa Liang, *Senior Member, IEEE*, and Jing Li

***Abstract*—With the integration of Mobile Edge Computing (MEC) and Network Function Virtualization (NFV), service providers are able to provide low-latency services to mobile users for profit. In this paper, we study the online service placement and request assignment problem in an MEC network, where service requests arrive one by one without the knowledge of future arrivals, and each arrived request demands a specific service with a tolerable service delay requirement with the aim to maximize the profit of the service provider, through admitting as many service requests as possible for a given monitoring period. This optimization objective is achieved by assigning service requests to appropriate cloudlets in the MEC network, pre-installing service instances into cloudlets to shorten service delays, and accommodating new services by revoking some idle service instances from cloudlets due to limited computing resources in MEC networks. In this paper, we first show that the problem is NP-hard. We then devise an efficient deep reinforcement learning algorithm for the online service placement and request assignment problem that consists of a deep reinforcement learning-based prediction mechanism for dynamic service placement, followed by a dynamic request assignment procedure to assign requests to cloudlets. We finally evaluate the performance of the proposed algorithms by conducting experiments through simulations. Simulation results demonstrate that the proposed algorithm is promising, improving performance by 46.8% compared with that of the comparison algorithms.**

***Index Terms*—Mobile edge-cloud networks, distributed resource allocation, service request provisioning, service instance placement, profit maximization, online machine-learning algorithms.**

I. Introduction

MOBILE edge computing (MEC) has been envisioned as a promising solution to provide adequate computing resources with high Quality of Service (QoS) through deploying cloudlets (edge servers) within the proximity of users [21]. Users can offload their delay-sensitive tasks to cloudlets instead of remote clouds for services, thereby reducing not only end-to-end service delays but also the communication resource consumption of core networks [37]. Combining MEC with virtualization techniques such as Network Function Virtualization (NFV), service providers now can rent out cloudlet resources for users by implementing user demanded services as service instances for profits [16]. For example, Augmented Reality (AR) devices are usually resource-constrained, AR service providers can place service instances at cloudlets to assist the processing of AR programs with low latency. On the other hand, AI services can be provided at cloudlets, where smart turnstiles can upload photos of visitors to MEC in order to verify visitor identities. Through these examples, it can be seen that different applications request different services. Meanwhile, the locations and arriving rates of different service requests are correlated, which usually follow spatial-temporal correlations. For instance, AR services are frequently requested from the locations such as museums or memorial halls while smart turnstiles usually request face identification services in rush hours.

When a user requests a service, its service provider will admit the request and initialize a service instance for the user. Once the requested service finishes, the service instance will be released back to the system to reduce the operational cost of the service provider, and the released resources can be utilized by future service requests. Most existing studies focused on how to assign service requests to different cloudlets in an MEC network, by either instantiating new service instances or sharing existing service instances [18], [19], [30], [35]. In contrast, in this paper we consider a scenario where the service provider of an MEC network can pre-install most demanded service instances in cloudlets in advance. The pre-installed service instances can be used by later service requests immediately without service initialization delays, this can shorten service delays of service requests. Consequently, the service provider can earn more profits by admitting more service requests while satisfying their service delay requirements. Since the service provider can only place limited service instances at resource-constrained cloudlets, an effective prediction mechanism that can predict which types of service requests and how many of the service instances are needed in a foreseeable future. Based on the service demand prediction, the system then decides whether to pre-install the predicted service instances or retain existing service instances for future use. Meanwhile, some existing service instances that are very unlikely to be used in the near future can be removed from the system to leave room for new service instances. In this paper, we will develop such a prediction mechanism to predict future service requests and their requested services. We will also devise an efficient algorithm for service instance placements built upon the prediction results, and we finally will perform service

Manuscript received 8 October 2021; revised 23 January 2022; accepted 13 March 2022. Date of publication 16 March 2022; date of current version 12 October 2022. The associate editor coordinating the review of this article and approving it for publication was M. Shojafar. *(Corresponding author: Weifa Liang.)*

Yuchen Li and Jing Li are with the School of Computing, The Australian National University, Canberra, ACT 2601, Australia (e-mail: yuchen.li@anu.edu.au; jing.li5@anu.edu.au).

Weifa Liang is with the Department of Computer Science, City University of Hong Kong, Hong Kong (e-mail: weifa.liang@cityu.edu.hk).

Digital Object Identifier 10.1109/TNSM.2022.3159744

request assignments to cloudlets to maximize the profit of the service provider. To this end, it poses several challenges.

- First, we are only allowed to install a limited number of service instances at each cloudlet. Since different services incur different operational costs and different amounts of computational resource consumption, it is challenging to decide how many service instances of each type of service should be installed at cloudlets in order to maximize the profit of the service provider. On the other hand, although the removal of an idle NFV instance can reduce the operational cost of the service provider, determining the removal of which idle NFV instances is challenging, as this can be illustrated by an extreme example, where newly incoming requests demand just removed service instances of a service, these newly arrived requests may not be admitted due to long instantiation delays on their demanded service instances. Also, the instance instantiation cost must be considered if it frequently happens to install and remove service instances from the system.
- Second, predicting future service types and service demands is difficult. Inaccurate prediction incurs a large initialization cost and unnecessary computing resource consumption. Therefore, without the knowledge of future service request arrivals and service types, judiciously placing different types of service instances to resource-limited cloudlets is challenging.
- Finally, it is challenging to assign service requests to different cloudlets in order to maximize the number of requests admitted, by taking into account service delay requirements of these requests, capacities of cloudlets, and the accumulative profit of admitted requests.

The novelty of this paper lies in a framework for dynamic service placement, by pre-installing service instances and revoking idle service instances through an effective prediction mechanism, with the aim to maximize the profit of the service provider. A deep-reinforcement learning-based online algorithm is devised through learning historical traces of service request patterns and service types at different cloudlets. A solution that can provide services for various requests then is delivered.

The main contributions of this paper are given as follows. We first formulate the online service placement and request assignment problem in MEC networks, by assigning service requests to cloudlets and pre-installing or removing service instances from cloudlets dynamically. This is achieved through analyzing historical service request patterns, subject to computing capacity on each cloudlet. We then develop an efficient algorithm for the problem, which consists of (i) a deep reinforcement learning prediction mechanism that is formulated as a Multi-agent Markov decision process to predict which services arrive in the future, and pre-install the demanded service instances accordingly; and (ii) service request assignment that assigns service requests to different cloudlets. We finally conduct experiments to evaluate the performance of the proposed algorithm. Experimental results demonstrate that the proposed algorithm is promising, and outperforms its comparison heuristics.

The rest of the paper is organized as follows. Section III introduces the system model and problem definition. Section IV develops an efficient algorithm for dynamic service request admissions that consist of a deep reinforcement learning-based prediction mechanism to predict different types of service request arrivals in order to install their service instances in the system, and an online algorithm for request assignment based on the installed service instances in cloudlets. Section V evaluates the performance of the proposed algorithms empirically, and Section VI concludes the paper.

II. Related Work

Several studies on service provisioning and Virtual Network Function (VNF) instance placement in MEC environments have been conducted in the past. For example, Xu *et al.* [39] jointly considered the placement of VNF instances for data processing and data traffic routing path planning for user requests in a multi-tier edge cloud network to maximize network throughput. They devised an efficient algorithm for request admissions, and an approximation algorithm for the problem without end-to-end delay requirements. Meanwhile, they dealt with the mobility of mobile devices by adopting machine-learning methods. Sun *et al.* [32] considered placing VNFs of Service Function Chains (SFC) to cloudlets and connecting VNFs in the SFC in order, with the aim to minimize the total placement cost. They proposed a time-efficient algorithm based on affiliation-aware VNF placement for an offline-version of the problem. They also utilized the Fourier-Series-based prediction method to predict the demands of VNFs for the online-version of the problem. Cziva *et al.* [6] studied the dynamic VNF placement problem to minimize the end-to-end latency of requests, considering network dynamics, user resource demands and user mobility. They developed a scheduler that delivers an optimal solution based on the optimal stopping theory [2]. Kayal and Liebeherr [14] considered service placement in a fog network to minimize the energy consumption and communication cost. They devised a fully distributed service placement algorithm based on the game theory. Li *et al.* [20] optimized the placement of SFCs to maximize the total expected profit under an assumption that both demanded computing resource and traffic data rates are uncertain. They proposed a near-optimal approximation algorithm by adopting the Markov approximation technique. He *et al.* [9] focused on the service placement and request scheduling while considering sharable and non-shareable resources in an MEC network with the aim to maximize the number of requests admitted. Yousefpour *et al.* [41] introduced a novel framework for QoS-aware dynamic fog service provisioning, where services can be dynamically deployed or released in fog nodes. They proposed two efficient greedy algorithms to minimize the cost while meeting the QoS requirements of applications. Hassan *et al.* [8] studied the service placement problem with the aim to reduce the response time for critical services and the energy consumption for non-critical services. They proposed an efficient service placement algorithm that considers the priority of services, network latency, and the power efficiency jointly. Ning *et al.* [29] studied the dynamic service placement framework where mobile users move erratically, with the aim to maximize the system utility. They first decomposed the long-term optimization problem by utilizing the Lyapunov optimization method. They then adopted a sample average approximation-based stochastic algorithm to approximate the future expected system utility. They finally determine service placement configurations by giving a distributed Markov-based approximation algorithm.

There are also investigations on the online service placement and request assignment problem of service provisioning in MEC networks. For example, Du *et al.* [7] investigated the problem of request offloading and content caching. They formulated the problem as a stochastic optimization problem with the aim of maximizing the profit, by jointly optimizing the computation offloading strategy, resource allocation, and Internet content caching strategy. They formulated a Lyapunov optimization algorithm for the problem. Li *et al.* [17] considered a VNF provisioning problem to maximize the revenue, by admitting user requests with service reliability requirements. They proposed two online algorithms with provable competitive ratios for the problem, by adopting the primal-dual technique. Yang *et al.* [40] aimed at maximizing the profit of cloud providers by admitting a subset of requests and finding an optimal scheduling of user requests. Ma *et al.* [26] studied the profit maximization problem in an MEC network by dynamically admitting delay-aware requests while meeting SFC requirements. Samanta and Chang *et al.* [31] focused on the optimization of the profit and latency. They proposed an adaptive service offloading scheme that maximizes the revenue collected while maintaining network utility. Xu *et al.* [38] considered the assignment of requests to shared VNF instances to maximize network throughput while minimizing the operational cost. They proposed a prediction mechanism to dynamically adjust the number of VNF instances needed in cloudlets, which later serves as a baseline in our experimental section. Liu *et al.* [23] studied the optimization of both new and existing user SFC deployments with the aim to maximize the profit, while considering th resource consumption and operational overhead. They proposed an integer linear programming (ILP) solution, and presented a column generation (CG) solution to reduce the time complexity of the algorithm.

There are several recent studies that utilize deep learning for task offloading and resource allocation problems in mobile edge computing networks [10], [24]. For instance, Tang and Wong *et al.* [34] considered task offloading in a mobile edge computing environment, where they utilized the long short-term memory (LSTM), dueling deep Q-network (DQN), and double-DQN techniques to decide whether to offload tasks, and to which edge node each offloading task should be offloaded. Huang *et al.* [12] considered task offloading and wireless resource allocation in an MEC under dynamic wireless network conditions. They proposed a Deep Reinforcement learning-based Online Offloading (DROO) framework, which achieves near-optimal performance with low computation time. Liu *et al.* [25] make use of deep reinforcement learning method to predict the locations of servers, and obtain an optimal task offloading decision in a Vehicle Edge Computing, where moving vehicles act as mobile edge servers to provide computation services. Min *et al.* [28] studied task offloading of IoT devices with energy harvesting. They proposed a reinforcement learning based scheme to decide the destination edge device of task offloading and the offloading rate. Chen *et al.* [3] proposed a double deep Q-network (DQN)-based strategic computation offloading algorithm to find the optimal task offloading destination base station, where the deep neural networks can predict the network dynamics in prior. However, the above studies mainly use deep learning as a solution to combinatorial optimization problems, or they utilize deep learning to predict the statuses of a network, such as the network conditions or the workload of servers.

Unlike the aforementioned studies that determine service placements when service requests are given in advance, in this paper we focus on placing service instances prior to service request arrivals, which is much more difficult as there is no way to know when and how many requests of each type of services arrive in advance. We will adopt the deep reinforcement learning method as a prediction mechanism to achieve this. It must be mentioned that this is an extended version of a recent conference paper [22].

III. Preliminaries

In this section, we introduce the system model and define the problem precisely.

A. System Model

We consider a mobile edge computing network $\mathcal{G} = (\mathcal{V} \cup S, E)$ that consists of a set $\mathcal{V}$ of access points (APs) and a set E of links between APs. For each link $e(i,j) \in E$, there is a transmission delay l_e $(= l_{i,j})$ on it. We assume that a link between two APs is an optical link with high capacity, and thus the bandwidth on the link is not considered as a constraint. Also, a set S of cloudlets that are co-located with some of the APs. Each cloudlet $c_j \in S$ has limited computing capacity CAP_j for hosting service instances. Let $f_k \in F$ with $k = 1, 2, \ldots, |F|$ be a service function in the service set F provided by the MEC network. We assume that a service instance of f_k can only serve a request at any moment, and thus, the computing resource requirement of a service instance can be precisely estimated in advance. Denote by $c(f_k)$ the computation resource requirement of service instance f_k. We further assume that the monitoring time period T is divided into equal time slots indexed by t with $1 \le t \le T$. In order to reduce the end-to-end service delay of each service request, service instances can be pre-installed in cloudlets prior to the arrivals of requests. Meanwhile, some idle service instances can also be removed from the system to deploy the most demanded services. Therefore, each time slot is split into two stages: *the service placement* stage, and *the request admission* stage. Fig. 1 demonstrates an MEC network with five APs and three co-located cloudlets, where several service instances are installed at cloudlet 1 and cloudlet 3, respectively.

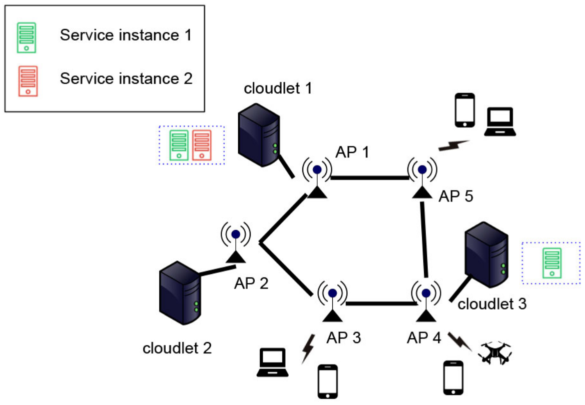


Fig. 1. An illustration of an MEC network with installed service instances.

B. Service Placement Stage

Denote by $n^t_{j,k}$ the number of instances of service f_k in cloudlet c_j at time slot *t* with $1 \le t \le T$. Since each cloudlet has limited computing capacity, the number of service instances installed in cloudlet c_j at time slot *t* is constrained by $\sum_{k=1}^{|F|} n^t_{j,k} \le CAP_j$, where CAP_j is the computing capacity of c_j. As service instances in cloudlets at each time slot *t* can be added or removed, let $\delta^t_{j,k}$ be the number of service instances to be installed or deleted from cloudlet c_j at time slot *t*. If $\delta^t_{j,k} > 0$, then cloudlet c_j instantiates $\delta^t_{j,k}$ instances of service f_k; otherwise, $\delta^t_{j,k}$ instances of f_k will be released from c_j. Furthermore, considering that some instances are serving unfinished requests, they should not be removed. Thus, the number of instances at each cloudlet should be no less than the number of unfinished requests at the cloudlet. For convenience, denote by $\phi^t_{j,k}$ the number of requests that are still being processed at time slot *t*, we have $\phi^t_{j,k} \le n^t_{j,k}$.

C. Request Admission Stage

Mobile users offload their tasks to cloudlets via their nearby APs. Let *R* be the set of service requests arrived at different APs during a monitoring period of *T*. We assume that at time slot *t*, a subset of requests *R*(*t*) ($\subset R$) arrives. Request $r_i \in R$ can be represented by a tuple $(t_i, f_{k_i}, v_i, \tau_i, d_i)$, where t_i is the time slot of its arrival; $f_{k_i} \in F$ represents its service type, $v_i \in \mathcal{V}$ represents its closest AP, τ_i represents its execution duration, and d_i is its end-to-end service delay requirement. For convenience, denote by $\mathcal{F}(k)$ ($\subset R$) the set of requests that demand service f_k.

Given an arrived request r_i, it will be either admitted by assigning it to one cloudlet or rejected immediately. We use a binary variable $x^t_{i,j} \in \{0, 1\}$ to denote whether request r_i is admitted by cloudlet c_j at time slot *t*. Request r_i can only be admitted at the time slot of its arrival,

$$x^t_{i,j} = 0, \quad \forall i, j, t \neq t_i \tag{1}$$

Because each service request r_i has an execution duration of τ_i time slots, once it is admitted, the request leaves the MEC system at the end of the $(t_i + \tau_i - 1)th$ time slot. We use $P^t_{i,j} \in \{0, 1\}$ to denote that request r_i is being processed by cloudlet c_j at time slot *t*, and thus

$$\forall i, j, t \quad P^t_{i,j} = \begin{cases} x^{t_i}_{i,j}, & t_i \le t < t_i + \tau_i \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

To admit request r_i, a cloudlet c_j needs to have an idle service instance or sufficient resource to initialize a new service instance for f_{k_i}. For convenience, we use a binary variable $x^{idle}_{t,i,j} \in \{0, 1\}$ to indicate if r_i is offloaded to an idle instance, and $x^{t^{new}}_{i,j} \in \{0, 1\}$ to indicate if request r_i is served by initializing a new instance. We then have

$$x^t_{i,j} = x^{idle}_{t,i,j} + x^{new}_{t,i,j}, \quad \forall i, j, t \tag{3}$$

Recall that $\phi^t_{j,k}$ is the number of unfinished requests for service f_k at the service placement stage of time slot *t*, we have

$$\phi^t_{j,k} = \sum_{r_i \in \mathcal{F}(k) \cap R(t)} P^t_{i,j}, \quad \forall t, j, k \tag{4}$$

Also, if there are $n^t_{j,k}$ instances for service f_k in cloudlet c_j at time slot *t* before the service admission stage of time slot *t* and $x^{new}_{t,i,j}$ instances are created at the request admission stage, the number of instances of service f_k at time slot $t + 1$ then is

$$n^{t+1}_{j,k} = n^t_{j,k} + \sum_{r_i \in \mathcal{F}(k)} x^{new}_{t,i,j} + \delta^{t+1}_{j,k}, \quad \forall t, j, k \tag{5}$$

To admit a request, not only does the cloudlet have sufficient residual computation capacity, but also the delay requirement d_i of request r_i can be met. Assigning request r_i to cloudlet c_j will experience the network delay, potential initialization delay, and processing delay. Denote by $proc_k$ the processing delay of an instance for service f_k, and define the network delay as the accumulative delay along a shortest path from the arrived AP to the target cloudlet, where a target cloudlet of a request is referred to its assignment to that cloudlet. Recall that link $e \in E$ between two APs has a latency l_e. The network delay of request r_i from an AP v_i to cloudlet c_j is $L_{i,j} = \sum_{e \in p_{i,j}} l_e$, where $p_{i,j}$ is the shortest path in the MEC network between AP v_i and AP v_j. We assume that the initialization time of service f_k is ins_k. If request r_i is uploaded to cloudlet c_j, and c_j does not have an idle instance for f_{k_i}, the cloudlet will instantiate a new instance for f_k. This will result in an initialization delay. The latency of assigning request r_i to cloudlet c_j thus is $L_{i,j} + ins_{k_i}$, which should be no greater than the delay requirement of request r_i, i.e.,

$$\left(L_{i,j} + proc_{k_i}\right) \cdot x^t_{i,j} + x^{new}_{t,i,j} \cdot ins_{k_i} \le d_i, \quad \forall i, j, t. \tag{6}$$

D. Operational Expense and Request Admission Profit

Initializing and hosting a service instance at any cloudlet incurs an operational cost. The operational expenditure of an MEC network consists of three components: (i) the energy consumption of hosting service instances even if the service instances are idle; (ii) the energy consumption of serving each service request; and (iii) the initialization cost of service instances.

Let $\mathcal{C}^{idle}(f_k)$ be the energy consumption per time slot for hosting an instance of service f_k, $\mathcal{C}^{ser}(f_k)$ the energy cost per time slot for admitting a request of service f_k, and $\mathcal{C}^{ins}(f_k)$ the initialization cost of an instance of service f_k. The operational cost $\mathcal{C}_j(t)$ of cloudlet c_j at time slot *t* thus is

$$\mathcal{C}_j(t) = \sum_{k=1}^{|F|}\Bigg(\Bigg(\sum_{r_i\in R(t)\cap\mathcal{F}(k)} x_{t,i,j}^{new} + \left[\delta_{j,k}^t\right]^+\Bigg)\cdot\mathcal{C}^{ins}(f_k) + \Bigg(\sum_{r_i\in R(t)\cap\mathcal{F}(k)} x_{t,i,j}^{new} + n_{j,k}^t\Bigg)\cdot\mathcal{C}^{idle}(f_k) + \sum_{r_i\in R(t)\cap\mathcal{F}(k)} P_{i,j}^t\cdot\mathcal{C}^{ser}(f_k)\Bigg), \quad (7)$$

where $[a]^+ = \max\{a, 0\}$, the first term in Eq (7) is the cost of initializing new instances, the second term is the idle energy consumption cost of hosting instances, and the third term is the energy consumption of serving requests.

The service provider charges users for their service requests are admitted. We assume that the service provider will earn the amount $RV(f_k)$ of revenues per time slot for serving a request of service f_k. Then, the total revenue obtained by cloudlet c_j at time slot *t* is

$$RV_j(t) = \sum_{r_i\in R(t)} \tau_i\cdot RV\big(f_{k_i}\big). \quad (8)$$

The profit of cloudlet c_j at time slot *t* is $RV_j(t) - \mathcal{C}_j(t)$. The total profit of the service provider by admitting user service requests for a given monitoring period *T* thus is

$$\sum_{t=1}^{T}\sum_{j=1}^{|S|}\big(RV_j(t) - \mathcal{C}_j(t)\big). \quad (9)$$

E. Problem Formulation

The *online service placement and request assignment* problem is defined as follows. Given a mobile edge-cloud network $\mathcal{G} = (\mathcal{V}\cup S; E)$, a set *F* of services provided by the network, a finite time horizon that is divided into *T* equal time slots, and a sequence of service requests arriving one by one without the knowledge of future arrivals, let *R* be the set of arrived requests within the given time horizon *T*, where each request $r_i \in R$ demands a service $f_{k_i} \in F$ with a delay requirement d_i. The problem is to assign arrived requests at each time slot *t* to cloudlets while meeting the delay requirements of admitted requests through (i) pre-deploying service instances of services in *F* to different cloudlets and determining the number of demanded service instances to meet the service delays of admitted requests; and (ii) removing some existing idle service instances from the system such that the accumulative profit $\sum_{j=1}^{|S|}\sum_{t=1}^{T}(RV_j(t) - \mathcal{C}_j(t))$ is maximized, subject to computing capacity on each cloudlet in $\mathcal{G}$.

Theorem 1: The online service placement and request assignment problem in $\mathcal{G} = (\mathcal{V}\cup S, E)$ is NP-hard.

Proof: We prove that the one-time-slot service placement problem with the knowledge of arrived requests is NP-hard, by a reduction from a well-known NP-hard problem - the multi-knapsack problem as follows.

Given a set *B* of knapsacks, where each knapsack $b_j \in B$ has capacity $W(b_j)$, and a set *A* of items where each item $a_i \in A$ has value $value(a_i)$ and size $size(a_i)$. The multi-knapsack problem is to maximize the accumulative value by placing as many items as possible into knapsacks, subject to the capacity on each knapsack.

Given an instance of a multi-knapsack problem, we construct a special case of the online service placement and request assignment problem as follows. For each knapsack b_j, we create a cloudlet c_j with capacity $W(b_j)$, and there is no idle service instance at the cloudlet. For each item a_i, we construct a request r_i with one-time-slot duration and the profit to admit it is $value(a_i)$, and the service instance f_{k_i} needs $size(a_i)$ computation capacity. Here we assume that the delay requirement is stringent that the requests can be admitted only if their required service instances are ready, and we also assume that the network delay is negligible so there is no difference among cloudlets when admitting requests. We aim to find how many instances to be installed at cloudlets while the computing capacity on each cloudlet is constrained. It can be seen that a solution to this special service placement and request assignment problem is a solution to the multi-knapsack problem, and the reduction is polynomial. Moreover, the online service placement and request assignment problem needs to place the instances without knowing the arrived requests at the current time slot, considering the impact of placement on future arrived requests, which is at least as hard as the special service placement and request assignment problem. Therefore, the problem is NP-hard. ■

F. ILP Formulation of the Offline Version of the Problem

In the following, we formulate an ILP formulation for the offline service placement and request assignment problem, where the set *R*(*t*) of requests at each time slot *t* is given with $1 \le t \le T$, i.e., $R = \cup_{t=1}^{T} R(t)$. For this offline version, the objective is to maximize the accumulative profit as follows.

$$\underset{x_{t,i,j}^{new}, x_{t,i,j}^{idle}, \delta_{j,k}^t}{\text{Maximize}} \sum_{t=1}^{T}\sum_{j=1}^{|S|}\big(RV_j(t) - \mathcal{C}_j(t)\big)$$

$$s.t. \quad (1), (2), (3), (4), (5), (6), (7), (8), (9) \quad (10)$$

$$\sum_{k=1}^{|F|} n_{j,k}^t\cdot c(f_k) \le CAP_j, \quad \forall j, t \quad (11)$$

$$\sum_{k=1}^{|F|} n_{j,k}^t\cdot c(f_k) + \sum_{i\in R(t)} x_{t,i,j}^{new}\cdot c\big(f_{k_i}\big) \le CAP_j, \quad \forall j, t \quad (12)$$

$$\sum_{i\in\mathcal{F}(k)} x_{t,i,j}^{idle} \le n_{j,k}^t - \phi_{j,k}^t, \quad \forall j, k, t \quad (13)$$

$$\phi_{j,k}^t \le n_{j,k}^t, \quad \forall j, k, t \quad (14)$$

$$\sum_{t=1}^{T}\sum_{j=1}^{|S|} x_{i,j}^{t} \le 1, \quad \forall i \tag{15}$$

$$n_{j,k}^{t} \ge 0, \quad \forall j,k,t \tag{16}$$

$$\delta_{j,k}^{t} \in Z, \quad \forall j,k,t \tag{17}$$

$$x_{i,j}^{t} \in \{0,1\}, \quad \forall i,j,t \tag{18}$$

$$x_{t,i,j}^{idle} \in \{0,1\}, \quad \forall i,j,t \tag{19}$$

$$x_{t,i,j}^{new} \in \{0,1\}, \quad \forall i,j,t \tag{20}$$

Constraint (11) guarantees that the total computing resource demand of all instances in the service placement stage does not exceed the computing capacity on each cloudlet at any time slot. Constraint (12) prevents cloudlets from admitting too many requests to violate their computation capacity. The RHS of Constraint (13) denotes the number of idle service instances. The LHS in Constraint (13) represents the number of requests for service f_k admitted by idle instances at cloudlet c_j, which should be no larger than the RHS. Constraint (14) ensures that the number of service instances is no less than the number of instances that are being processed, so in service placement stage we do not remove non-idle service instances. Constraint (15) ensures that each request can only be admitted by one cloudlet.

Although the ILP formulation of the offline version of the problem can deliver an optimal solution, this solution is an upper bound on the problem due to that it is assumed that the knowledge of request arrivals at all time slots is given. The LP relaxation of the proposed ILP solution delivers a solution whose value is an upper bound on the value of the optimal solution of the offline version of the problem, which is also an upper bound on the problem. This relaxed solution thus will be used as an estimate of the optimal solution of the problem later.

For the sake of convenience, we list all symbols adopted in this paper in Table I.

IV. Online Algorithm

In this section we present an online algorithm for the online service instance placement and request assignment problem.

A. Algorithm Overview

The basic idea of the proposed algorithm consists of two stages: the service placement stage, followed by the request assignment stage. We first adopt a deep reinforcement learning method to predict service request demands and types of services. We then find a near-optimal service placement for the predicted service instances. We finally assign each incoming service request to a cloudlet that contains the service instance or instantiate an instance of the demanded service.

B. Request Assignment

Assuming that service instances have been placed in cloudlets, we deal with service request assignment with the aim to maximize the accumulative profit at each time slot *t* with $1 \le t \le T$. Given that a set *R*(*t*) of requests arrives and some service instances have already been deployed in cloudlets at time slot *t*, we reduce request assignment to the Generalized Assignment Problem (GAP) [4] to maximize the profit.

Given a one-time-slot request assignment, we treat each idle service instance of f_k as a bin with capacity $c(f_k)$. Also, each cloudlet is treated as a bin with its remaining computing capacity $CAP_j - \sum_{k=1}^{|F|} n_{j,k}^{t} \cdot c(f_k)$ at time slot *t*. Then, each request r_i corresponds to an item. For bins corresponding to idle service instances, if the delay requirement of a request can be satisfied when assigned to the bin, its size is set as $c(f_{k_i})$, and the profit is set as $(RV(f_{k_i}) - \mathcal{C}^{ser}(f_{k_i})) \cdot \tau_i$. Similarly, for bins corresponding to the residual computing capacities of cloudlets, the size of an item at the bins is $c(f_{k_i})$, and the profit is set as $(RV(f_{k_i}) - \mathcal{C}^{ser}(f_{k_i}) - \mathcal{C}^{idle}(f_{k_i})) \cdot \tau_i - \mathcal{C}^{ins}(f_{k_i})$ due to the extra cost introduced by initializing new instances. For idle service instances or cloudlets that cannot fulfil the delay requirement of request r_i, we set its size as ∞ to indicate that r_i cannot be admitted. It can be seen that an approximate solution to the GAP in turn returns an approximate solution to the request assignment problem. The detailed procedure is given in `Procedure` 1.

C. Service Placement

Considering that no prior knowledge of request arrivals at each time slot *t* is given, we adopt a deep reinforcement learning method for service placement. Specifically, we first formulate the service placement as a Multi-agent Markov Decision Process (MMDP), where a set of agents interacts with an environment for a given number of time slots, with the aim to maximize the total reward. At each time slot, the agents obtain the state of the environment, and then take actions based on the state. The actions then impact the environment and transit the environment to the next state. The agents receive rewards as a consequence of their actions and the change of the environment. Fig. 2 demonstrates the interaction of agents and their environments.

An MMDP is usually expressed by a tuple $(\mathcal{N}, \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, where $\mathcal{N}$ is the set of agents, $\mathcal{S}$ is the state set of the environment, $\mathcal{A}$ is the action set of agents, $\mathcal{P}$ is the transition function, and $\mathcal{R}$ is the reward function. Given state s, actions $a_1, a_2, \ldots, a_{|\mathcal{N}|}$ and the next state s', $\mathcal{P}(s, a_1, a_2, \ldots, a_{|\mathcal{N}|}, s')$ defines the probability of the transition from state s to s'. Similarly, $\mathcal{R}(s, a_1, a_2, \ldots, a_{|\mathcal{N}|}, s')$ is the reward that the agents obtain by taking action a when the state transits from s to s'.

We formulate the problem as a multi-agent Markov Decision Process (MMDP) as follows.

Agent: Each cloudlet c_j with $j \in \{1, 2, \ldots, |S|\}$ is modeled by an agent $\mathcal{N}_j$ to manage its service placement.

State: The state $s \in \mathcal{S}$ describes the status of the MEC network, i.e., the history of offloaded service requests that will be admitted at cloudlets and their demanded service instances in cloudlets. Thus, the system state $s^t \in \mathcal{S}$ at time slot *t* is defined by a tuple $(R_\eta(t), n(t), R^*(t), t)$, where $R_\eta(t) = R(1) \cup R(2) \cup \ldots \cup R(t-1)$ is the set of service requests that arrive before time slot *t*, $n_{j,k}^{t} \in n(t)$ is the number of instances of service *k* at cloudlet c_j, $R^*(t)$ denotes the set

TABLE I
TABLE OF SYMBOLS

Notations	Descriptions
$\mathcal{G} = (\mathcal{V} \cup S, E)$	an MEC network with a set V of APs, a set S of cloudlets attached to the APs, and a set of E links
l_e and $l_{(i,j)}$	the transmission delay on edge e between AP v_i and v_j
c_j and CAP_j	a cloudlet $c_j \in S$ and the computing capacity of c_j
F,f_k, $c(f_k)$, and ins_k	a set of service provided in the MEC network, a service $f_k \in F$, the occupied computing capacity and initialization time of a service instance for f_k
t and T	the time slot index of monitoring period and the maximum time slot
$n^t_{j,k}$	the number of instances of service f_k at cloudlet c_j after the service placement stage at time slot t
$\delta^t_{j,k}$	a non-negative integer decision variable on the number of instances to be installed or deleted at the service placement stage of time slot t
$\phi^t_{j,k}$	the number of requests that are being processed at time slot t and arrive before time slot t
R and $R(t)$	the set of service requests that arrive during the monitoring period and the set of service requests that arrive at time slot t
$\mathcal{F}(k)$	the set of requests that demand service f_k
r_i, t_i,f_{k_i},v_i,τ_i, and d_i	Request $r_i \in R$ and its arriving time, demanded service, arrived AP, execution duration and end-to-end delay requirement
$x^t_{i,j}$	a binary variable indicating whether request r_i is admitted by cloudlet c_j at time slot t or not
$x^{new}_{t,i,j}$	a binary decision variable indicating whether request r_i is offloaded to a newly initialized instance at cloudlet c_j or not
$x^{idle}_{t,i,j}$	a binary decision variable indicating whether request r_i is offloaded to an idle instance at cloudlet c_j or not
$P^t_{i,j}$	a binary variable indicating whether request i is being processed by cloudlet c_j at time slot t
$p_{i,j}$	the shortest path in the MEC network between AP v_i and AP v_j
$L_{i,j}$	the network delay of offloading request r_i from v_i to cloudlet c_j
$\mathcal{C}^{idle}(f_k)$ and $\mathcal{C}^{ser}(f_k)$	the energy consumption per time slot for hosting an instance of service f_k and processing a request of service f_k
$\mathcal{C}^{ins}(f_k)$	the initialization cost for an instance of service f_k
$\mathcal{C}_j(t)$	The operational cost of cloudlet c_j at time slot t
$RV(f_k)$	revenues per time slot for processing a request of service f_k
$RV_j(t)$	the revenue of cloudlet c_j at time slot t
$\mathcal{N}$ and $\mathcal{N}_j$	a set of agents and the agent on cloudlet c_j
$\mathcal{S}$ and s	a set of states and a state $s \in \mathcal{S}$
$\mathcal{A}$ and a	a set of actions and an action $a \in \mathcal{A}$
$\mathcal{P}$ and $\mathcal{P}(s, a_1, a_2, \ldots, a_{\|\mathcal{N}\|}, s')$	A state transition function, and the probability of the transition from state s to s' after agent actions
$\mathcal{R}$ and $\mathcal{R}_j(s, a_1, a_2, \ldots, a_{\|\mathcal{N}\|}, s')$	A reward function, and the reward that the agent $\mathcal{N}_j$ obtains after taking action a_j if the state transits from s to s'
$R_\eta(t)$	$R_\eta(t) = R(1) \cup R(2) \cup \ldots \cup R(t-1)$ is the set of service requests that arrive before time slot t
$R^*(t)$	the set of requests that are being processed at cloudlets at time slot t
a'_j	an encoded action of agent $\mathcal{N}_j$
$a'_{j,0}$	percentage of available computational capacity to be allocated to service f_k
$cap^{rej}_{t,j}$	the available computational capacity at the service placement stage of time slot t
$G_j(t')$	the cumulative reward of agent $\mathcal{N}_j$ from time slot t' to T
θ_j	the learnable DNN parameters of actor networks of agent $\mathcal{N}_j$
π_j and $\pi_j(a_j \mid s; \theta_j)$	the policy function of agent $\mathcal{N}_j$ and the probability of agent $\mathcal{N}_j$ taking action a_j under system state s with DNN parameter
J^t_j	a function mapping parameter θ_j to expected cumulative rewards after time slot t of agent $\mathcal{N}_j$
$\nabla_{\theta_j} J^t_j(\theta_j)$	the gradient of $J^t_j(\theta_j)$
η	the learning rate of training
ω_j	the learnable DNN parameters of critic networks of agent $\mathcal{N}_j$
$V^{\pi_j}_j(s^t; \omega_j)$	a function that approximates $\mathbb{E}[G(t) \mid s^t]$

of admitted service requests prior to time slot *t*, that have not finished at time slot *t*.

Action: In the service placement stage of each time slot, the agents decide to place or remove some service instances at their cloudlets. Thus, action $a_j \in \mathcal{A}$ of agent $\mathcal{N}_j$ is defined as a vector $a_j = (a_{j,0}, a_{j,1}, a_{j,2}, \ldots, a_{j,|F|})$, $\sum_{k=0}^{|F|} a_{j,k} = 1$, where $a_{j,k}$ represents the percentage of available computational capacity to be allocated to service f_k if $k \neq 0$ and $a_{j,0}$ represents the percentage of available computational capacity. While deciding the placement of instances, we need to maintain the instances of unfinished requests. Therefore, the available computational capacity that

Procedure 1 Request Assignment Procedure at One Time Slot

```
Input: An MEC network $\mathcal{G} = (\mathcal{V} \cup S; E)$ with a set of cloudlets $C$, the number of total service instance $n^t_{j,k}$ and occupied service instance $\phi^t_{j,k}$ for service $f_k$ at each cloudlet $c_j$, and a set of requests $R(t)$ that arrives at time slot $t$.
Output: Find an assignment of requests in $R(t)$ at time slot $t$ so that the total profit is maximized, subject to the capacity of cloudlets and delay requirement.
1: Calculate the delay $L$ between any pairs of APs and APs co-located with cloudlets by the Floyd-Warshall algorithm [5];
2: Initialize an empty set of bins $\mathcal{B} \leftarrow \emptyset$;
3: for $j \leftarrow 1, 2, \dots, |S|$ do
4:     for $k \leftarrow 1, 2, \dots, |F|$ do
5:         for $b \leftarrow 1, 2, \dots, n^t_{j,k} - \phi^t_{j,k}$ do
6:             Create a new bin $B$ with capacity $c(f_k)$;
7:             $\mathcal{B} \leftarrow \mathcal{B} \cup \{B\}$ ;
8:         end for
9:     end for
10:    Create a new bin $B'$ with capacity $CAP_j - \sum_{k=1}^{|F|} n^t_{j,k} \cdot c(f_k)$;
11:    $\mathcal{B} \leftarrow \mathcal{B} \cup \{B'\}$;
12: end for
13: Initialize an empty set of items $\mathcal{I} \leftarrow \emptyset$;
14: for $i \leftarrow 1, 2, \dots, |R(t)|$ do
15:     Create a new item $I$ and $\mathcal{I} \leftarrow \mathcal{I} \cup \{I\}$;
16:     for $j' \leftarrow 1, 2, \dots, |\mathcal{B}|$ do
17:         if Bin $B'_j$ corresponds to an idle service instance at cloudlet $c_j$ then
18:             $Profit(i, j') \leftarrow (RV(f_{k_i}) - \mathcal{C}^{ser}(f_{k_i})) \cdot \tau_i$;
19:             $delay(i, j') \leftarrow L_{i,j}$;
20:         else
21:             $Profit(i, j') \leftarrow (RV(f_{k_i}) - \mathcal{C}^{ser}(f_{k_i}) - \mathcal{C}^{idle}(f_{k_i})) \cdot \tau_i - \mathcal{C}^{ins}(f_{k_i})$;
22:             $delay(i, j') \leftarrow L_{i,j} + ins_{k_i}$;
23:         end if
24:         if $delay(i, j') \le D_i$ then
25:             $weight(i, j') \leftarrow c(f_{k_i})$;
26:         else
27:             $weight(i, j') \leftarrow \infty$;
28:         end if
29:     end for
30: end for
31: Find an assignment of items $\mathcal{I}$ to $\mathcal{B}$ such that the accumulative profit of assigned items is maximized under the constraint that the total weight of items in each bin does not exceed its capacity, by invoking the approximation algorithm for the GAP in [4];
32: return The assignment of requests $x^{new}_{t,i,j}$ and $x^{idle}_{t,i,j}$ derived from the assignment of items.
```

can be allocated is $cap^{rej}_{t,j} = CAP_j - \sum_{k=1}^{|F|} \phi^t_{j,k} \cdot c(f_k)$. The action a_j can be decoded by $\delta^t_{j,k} = \lfloor \frac{cap^{rej}_{t,j} \cdot a'_{j,k}}{c(f_k)} \rfloor$ for service placement.

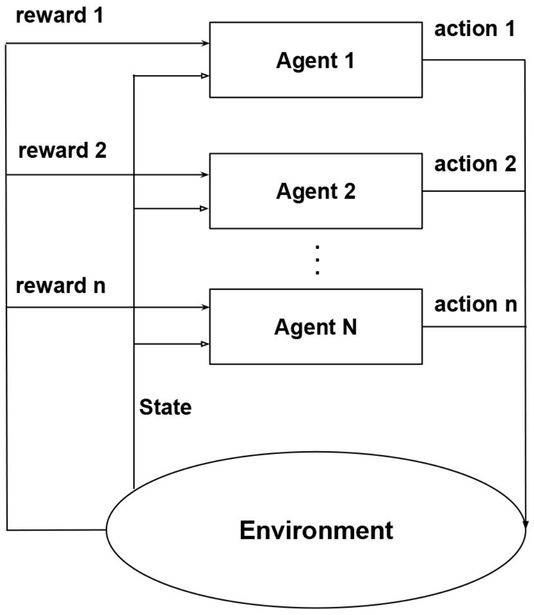


Fig. 2. An illustration of a Multi-agent Markov Decision Process in one time slot.

State Transition: Having taken agent actions, the system state will transit to the next state, following the state transition function $P(s^t, a^t_1, a^t_2, \dots, a^t_{|S|}, s^{t+1})$. Specifically, first, the number of instances at any cloudlet $c_j \in S$ changes under the agent's action a_j in the service placement stage, that is, $n^t_{j,k} \leftarrow n^t_{j,k} + \delta^t_{j,k}$. Second, in the request admission stage, a set of requests $R(t)$ arrives. Procedure 1 (that will be disscussed later) is applied to assign the arrived requests to cloudlets. The system state then is updated, following the result delivered by Procedure 1. Since finished requests will leave in the end of each time slot, set $R^*(t)$ will be updated accordingly.

Reward: To encourage each agent to maximize its contribution to the total profit, a reward function $\mathcal{R}_j(s^t, a^t_1, a^t_2, \dots, a^t_{|S|}, s^{t+1})$ is defined, which describes how much reward an agent $\mathcal{N}_j$ receives after a state transition. Assume that there is a state transition $s^t = (R_\eta(t), n(t), R^*(t), t) \xrightarrow{a^t_1, a^t_2, \dots, a^t_{|S|}} s^{t+1} = (R_\eta(t+1), n(t+1), R^*(t+1), t+1)$. Let $x^{new}_{t,i,j}$ and $x^{idle}_{t,i,j}$ be the outputs of Procedure 1 during the transition. The revenue and operation cost of cloudlet c_j are calculated by Eq. (7) and Eq. (8), respectively.

The reward function of agent $\mathcal{N}_j$ then is defined as follows.

$$\mathcal{R}_j\left(s^t, a^t_1, a^t_2, \dots, a^t_{|S|}, s^{t+1}\right) = RV_j(t) - \mathcal{C}_j(t). \quad (21)$$

The cumulative reward of agent $\mathcal{N}_j$ from time slot t' to T then is

$$G_j(t') = \sum_{t=t'}^{|T|} \mathcal{R}_j\left(s^t, a^t_1, a^t_2, \dots a^t_{|S|}, s^{t+1}\right). \quad (22)$$

It can be seen that $G_j(1)$ is the total reward of agent $\mathcal{N}_j$ during the monitoring period T, so $\sum_{j=1}^{|S|} G_j(1)$ is equivalent to the objective function (9). We thus maximize the total

Procedure 2 Advantage Actor-Critic Based Procedure for Online Service Instance Placement

```
Input: The system state $s^t = (R_\eta(t), n(t), R^*(t), t)$, the
    actor network with parameter $\theta_j$ for agent $\mathcal{N}_j$.
Output: The service placement decision for cloudlet $c_j$
1: Let $\pi_j(a_j^t \mid s^t; \theta_j)$ be the output of the actor network for
   the input of system state $s^t$;
2: Sample action $a_j^t$ according to the distribution
   $\pi_j(a_j^t \mid s^t; \theta_j)$;
3: $\delta_{j,k}^t \leftarrow \lfloor \frac{cap_{t,j}^{rej} \cdot a_{j,k}'^t}{c(f_k)} \rfloor$,   $\forall k \in \{1, 2, \ldots, |F|\}$;
4: return The service placement decision $\delta_{j,k}^t$ for cloudlet
   $c_j$
```

reward $G_j(1)$ of all agents $\mathcal{N}_j$ instead of maximizing the original optimization objective of the problem. Due to the large state space and lack of knowledge of future request arrivals, it is impossible to calculate the state transition function and reward function accurately. We instead design a policy function $\pi_j(a_j \mid s)$ for each agent $\mathcal{N}_j$, which gives the possibility of taking action a under state s. We then find an optimal policy function of each agent. We make use of deep neural networks (DNNs) called *actor networks* as the policy function $\pi_j(a_j \mid s)$, which can extract important features from states and learn service request patterns. The service placement by DNNs can be obtained by applying `Procedure 2` as follows.

Initially, each agent in the actor network is not trained and has a random parameter θ_j. The reward could be low. To obtain a near-optimal reward, assume that the service provider has a collection Γ of sets of historical requests in the past monitoring periods, where a set $R' \in \Gamma$ of requests arrives during one of the past monitoring periods which has an almost identical arriving pattern as R. We train the actor networks using historical data traces to approximate the optimal policy functions of agents as disscussed below.

D. Online Algorithm

Considering that there are multiple agents and the action space is continuous. To train the actor networks, we adopt the Advantage Actor-Critic (A2C) algorithm for multiple agents and continuous space scenario [36], where the actor networks of all agents have randomly initialized parameter θ_j. Denote by $\pi_j(a_j \mid s^t; \theta_j)$ the actor network with the parameter θ_j for each agent $\mathcal{N}_j$.

The basic idea of the A2C algorithm is to apply Stochastic gradient descent(SGD) technique [15] to update parameter θ_j in order to maximize the expected cumulative reward of agent $\mathcal{N}_j$. The expected cumulative reward of agent $\mathcal{N}_j$ from time slot 1 to T is defined as a function J_j^t with respect to parameter θ_j by

$$J_j(\theta_j) = \mathbb{E}\big[G_j(1); \theta_j\big]. \tag{23}$$

Since we aim to maximize $J_j(\theta_j)$, this is equal to minimizing $-J_j(\theta_j)$. We update parameters θ_j of the actor network by applying SGD, i.e.,

$$\theta_j \leftarrow \theta_j + \eta \nabla_{\theta_j} J_j(\theta_j), \tag{24}$$

where $\nabla_{\theta_j} J_j(\theta_j)$ is the gradient of $J_j(\theta_j)$, and η is the learning rate to control the step of the gradient descent, which is a constant with $0 < \eta < 1$.

Denote by $\zeta = (s^1, a_1^1, \ldots, a_{|S|}^1, s^2, \ldots, a_{|S|}^{|T|}, s^{|T|})$ a sequence of system states and agents' actions for the monitoring period T. Let $Pr(\zeta)$ be the probability of sequence ζ and $\mathcal{R}_j(\zeta)$ the reward that agent $\mathcal{N}_j$ obtained if the sequence of system states and agent actions follows ζ. Eq. (23) can be written as

$$J_j(\theta_j) = \int Pr(\zeta) \cdot \mathcal{R}_j(\zeta) d\zeta. \tag{25}$$

Note that for function $f(\cdot)$, we have $\nabla f(\cdot) = f(\cdot)\nabla \log f(\cdot)$. To calculate the gradient $\nabla_{\theta_j} J_j(\theta_j)$, we have

$$\begin{aligned} \nabla_{\theta_j} J_j(\theta_j) &= \nabla_{\theta_j}\left(\int Pr(\zeta) \cdot \mathcal{R}_j(\zeta)\ d\zeta\right) \\ &= \int \mathcal{R}_j(\zeta) \cdot \left(\nabla_{\theta_j} Pr(\zeta)\right)\ d\zeta \\ &= \int Pr(\zeta) \cdot \mathcal{R}_j(\zeta) \cdot \left(\nabla_{\theta_j} \log Pr(\zeta)\right)\ d\zeta \\ &= \mathbb{E}\Big[\mathcal{R}_j(\zeta) \cdot \left(\nabla_{\theta_j} \log Pr(\zeta)\right)\Big]. \end{aligned} \tag{26}$$

The probability $Pr(\zeta)$ can be expanded, that is,

$$\begin{aligned} Pr(\zeta) = p\left(s^1\right) \cdot \prod_{t=1}^{|T|}\Bigg(\mathcal{P}\left(s^t, a_1^t, \ldots, a_{|S|}^t, s^{t+1}\right) \\ \cdot \prod_{j'=1}^{|S|} \pi_j'\left(a_j' \mid s^t; \theta_j'\right)\Bigg), \end{aligned} \tag{27}$$

where $p(s^1)$ is the probability of the initial state s_1, $\prod_{t=1}^{|T|}(\mathcal{P}(s^t, a_1^t, \ldots, a_{|S|}^t, s^{t+1}) \cdot \prod_{j'=1}^{|S|} \pi_j'(a_j' \mid s^t; \theta_j'))$ is the probability that the system has a sequence of states $s_1, s_2, \ldots, s_{|T|}$, and each agent $\mathcal{N}_j$ takes actions $a_j^1, a_j^2, \ldots, a_j^{|T|}$ at different time slots.

Pluging Eq. (27) into Eq. (26), we have

$$\begin{aligned} \nabla_{\theta_j} J_j(\theta_j) = \mathbb{E}\Bigg[&\left(\sum_{t=1}^{T} \mathcal{R}_j\left(s^t, a_1^t, \ldots, a_{|S|}^t, s^{t+1}\right)\right) \\ &\cdot \Bigg(\nabla_{\theta_j} \log\ p\left(s^1\right) \\ &\cdot \prod_{t=1}^{|T|}\Bigg(\mathcal{P}\left(s^t, a_1^t, \ldots, a_{|S|}^t, s^{t+1}\right) \\ &\cdot \prod_{j'=1}^{|S|} \pi_j'\left(a_j' \mid s^t; \theta_j'\right)\Bigg)\Bigg)\Bigg] \\ = \mathbb{E}\Bigg[&\left(\sum_{t=1}^{T} \nabla_{\theta_j} \log \pi_j\left(a_j \mid s^t; \theta_j\right)\right) \\ &\cdot \left(\sum_{t=1}^{T} \mathcal{R}_j\left(s^t, a_1^t, \ldots, a_{|S|}^t, s^{t+1}\right)\right)\Bigg]. \end{aligned} \tag{28}$$

However, it is impossible to calculate the expectation Eq. (28) directly. Therefore, we instead sample historical service request traces to estimate $\nabla_{\theta_j} J_j(\theta_j)$. That is,

$$\nabla_{\theta_j} J_j(\theta_j) \approx \left(\sum_{t=1}^{T} \nabla_{\theta_j} \log \pi_j(a_j \mid s^t;\theta_j)\right) \cdot \left(\sum_{t=1}^{T} \mathcal{R}_j\left(s^t, a_1^t, \ldots, a_{|S|}^t, s^{t+1}\right)\right). \quad (29)$$

Notice that action a_t will affect the reward received prior to time slot *t*. Due to causality [33], we instead use the following estimation

$$\nabla_{\theta_j} J_j(\theta_j) \approx \sum_{t=1}^{T} \nabla_{\theta_j} \log \pi_j(a_j \mid s^t;\theta_j) \cdot \left(\sum_{t'=t}^{T} \mathcal{R}_j\left(s^t, a_1^t, \ldots, a_{|S|}^t, s^{t+1}\right)\right) = \sum_{t=1}^{T} \nabla_{\theta_j} \log \pi_j(a_j \mid s^t;\theta_j) \cdot G_j(t), \quad (30)$$

To further decrease the estimated variance, we replace $G_j(t)$ with $G_j(t) - \mathbb{E}[G_j(t) \mid s^t]$, and the details of the proof of the replacement can be found in [1], [33]. We then have

$$\nabla_{\theta_j} J_j(\theta_j) \approx \sum_{t=1}^{T} \nabla_{\theta_j} \log \pi_{\theta_j}\left(a_j^t \mid s^t;\theta_j\right)\left(G_j(t) - \mathbb{E}[G_j(t) \mid s^t]\right), \quad (31)$$

as an estimation of gradient $\nabla_{\theta_j} J_j(\theta_j)$, and we deploy a critic network for each agent that takes s^t as its input and a value function $V_j^{\pi_j}(s^t;\omega_j)$ as it soutput, which serves as an approximate estimation on $\mathbb{E}[G_j(t) \mid s^t]$.

Similarly, each critic network has a random parameter ω_j to be trained, we will train both the actor and critic networks of each agent $\mathcal{N}_j$ simultaneously. With the progress of training, the gap between $V_j^{\pi_j}(s^t;\omega_j)$ and $\mathbb{E}[G_j(t) \mid s^t]$ becomes smaller, which makes the actor network approaches the optimal policy.

The training proceeds as follows. Let R' be a set of historical service request traces. We simulate the service placement and request assignment processes, using historical service requests in R'.

In the service placement stage at each time slot *t*, each agent $\mathcal{N}_j$ first inputs the system state s^t into its actor network, its output $\pi_j(a_j \mid s^t;\theta_j^0)$ and service instance placement are obtained accordingly.

In the request assignment stage at time slot *t*, `Procedure` 1 is invoked to assign the arrived requests in $R'(t)$. The system state s^t, action a_j^t and reward $\mathcal{R}_j^t$ of each agent $\mathcal{N}_j$ at each time slot *t* are recorded. Then, the service placement stage at time slot $t+1$ proceeds. This procedure is repeated until the last time slot T reaches.

We then update the actor and critic networks of each agent $\mathcal{N}_j$. Recall that we aim to maximize $J_j^t(\theta_j)$ at each time slot *t*, and we recorded the system state s^t, action a_j^t and reward $\mathcal{R}_j^t$ of each agent $\mathcal{N}_j$ at each time slot *t*.

Procedure 3 Training Procedure for the Actor Network

```
Input: An MEC network $\mathcal{G} = (\mathcal{V} \cup S; E)$ with a set of
    cloudlets $C$, and a set of services $F$ provided by the
    network and historical requests in the monitoring area.
Output: A set of trained Actor networks for all agents
1:  for $j \leftarrow 1, 2, \ldots, |S|$ do
2:     Randomly initialize an actor network with parameter $\theta_j$
       and a critic network with parameter $\omega_j$ for each agent;
3:  end for
4:  while $\theta_j$ and $\omega_j$ has not converged do
5:     Select a set $R'$ of service requests from request history;
6:     for $t \leftarrow 1, 2, \ldots, T$ do
7:        for $j \leftarrow 1, 2, \ldots, |S|$ do
8:           Calculate $\delta_{j,k}^t$  $\forall k \in \{1, 2, \ldots, |F|\}$ by calling
             Procedure 2;
9:        end for
10:       In request admission stage, calculate $x_{t,i,j}^{new}$ and $x_{t,i,j}^{idle}$
          of requests in $R'(t)$ by invoking Procedure 1, and
          update the system according to the assignment.
11:       Calculate the rewards $G_j^t$ for each agent;
12:    end for
13:    for $j \leftarrow 1, 2, \ldots, |S|$ do
14:       for $t \leftarrow 1, 2, \ldots, T$ do
15:          Input system state $s^t$ into the critic networks;
16:          $\omega_j \leftarrow \omega_j - \eta\nabla_{\omega_j}(G_j(t) - V_j^{\pi_j}(s^t;\omega_j))^2$;
17:       end for
18:       $\theta_j \leftarrow \theta_j + \eta\sum_{t=1}^{T} \nabla_{\theta_j} \log \pi_{\theta_j}(a_j^t \mid s^t;\theta_j)(G_j(t) -$
          $V_j^{\pi_j}(s^t;\omega_j))$;
19:    end for
20: end while;
21: return The trained actor networks with parameters
       $\theta_1, \theta_2, \ldots, \theta_{|S|}$.
```

To minimize the difference between the output $V_j^{\pi_j}(s^t;\omega_j)$ and $\mathbb{E}[G(t)|s^t]$, we first train the critic network by updating ω_j, followed by updating θ_j, i.e.,

$$\omega_j \leftarrow \omega_j - \eta\nabla_{\omega_j}\left(G_j(t) - V_j^{\pi_j}(s^t;\omega_j)\right)^2, \quad (32)$$

and

$$\theta_j \leftarrow \theta_j + \eta\sum_{t=1}^{T} \nabla_{\theta_j} \log \pi_{\theta_j}\left(a_j^t \mid s^t;\theta_j\right) \times \left(G_j(t) - V_j^{\pi_j}(s^t;\omega_j)\right). \quad (33)$$

Having updated both θ_j and ω_j, if they still do not converge, we can select another set R' of historical requests from the collection Γ randomly, and repeat the above steps on R' to further train the actor network and critic network of each agent $\mathcal{N}_j$ until the convergence of both θ_j and ω_j. The detailed process is shown in `Procedure` 3.

Having trained the DNNs, the actor networks then can be used for service placement. As shown in Fig. 3, at the service

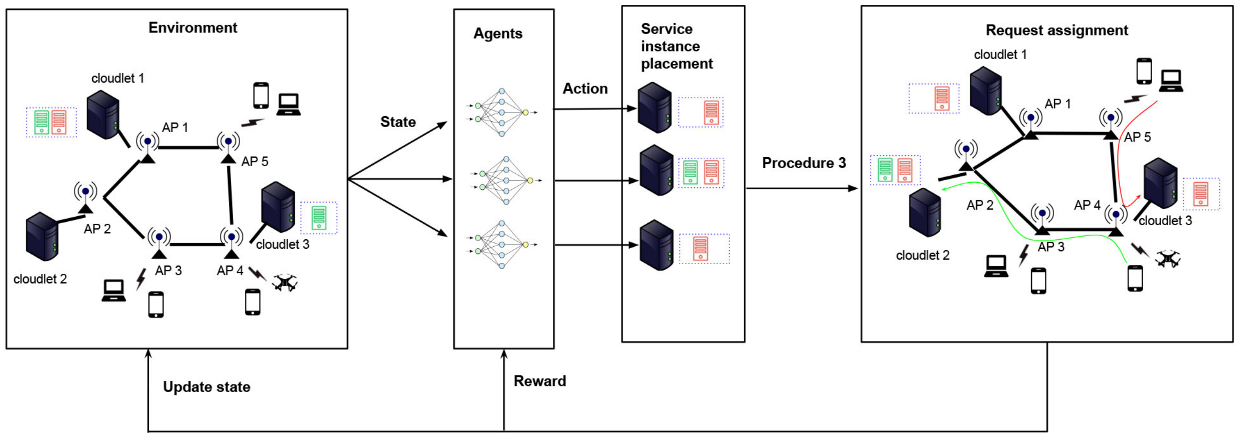


Fig. 3. An illustration of the service placement and request assignment procedure at time slot t.

Algorithm 1 Advantage Actor-Critic Based Algorithm for Online Service Placement and Request Assignment Problem

```
Input: An MEC network $\mathcal{G} = (\mathcal{V} \cup S; E)$ , and a set of
    services $F$ provided by the network, a set of requests $R$,
    and historical requests in the monitoring area.
Output: The service placement decision and request assign-
    ment at each time slot.
1: Train the actor networks by applying Procedure 3;
2: for $t \leftarrow 1, 2, \ldots, T$ do
3:     for $j \leftarrow 1, 2, \ldots, |S|$ do
4:         In service placement stage, call Procedure 2 to
           calculate $\delta^t_{j,k}$ , $\delta^t_{j,k}$ instances for service $f_k$ will be
           installed or deleted at Cloudlet $c_j$;
5:     end for
6:     In request admission stage, calculate $x^{new}_{t,i,j}$ and $x^{idle}_{t,i,j}$ of
       requests in $R(t)$ by invoking Procedure 1, and update
       the system according to the assignment;
7: end for
8: return The service placement decision $\delta^t_{j,k}$ derived from
   the actions $a^t_j$ and the request assignments $x^{new}_{t,i,j}$, $x^{idle}_{t,i,j}$.
```

placement stage of each time slot t, we apply Procedure 2, where the agents take actions, i.e., decide the service placement on cloudlets based on the state of the MEC network. In Fig. 3, cloudlet 1 removes service instance 1, cloudlet 2 installs service instances 1 and 2, and cloudlet 3 installs service instance 2. At the request admission stage, we apply Procedure 1 to assign requests to appropriate cloudlets. As shown in Fig. 3, the request that arrived at AP_4 is assigned to cloudlet 2, and the request that arrives at AP_5 is offloaded to cloudlet 3. We finally update the system state and proceed in the next time slot. The detailed algorithm is shown in Algorithm 1.

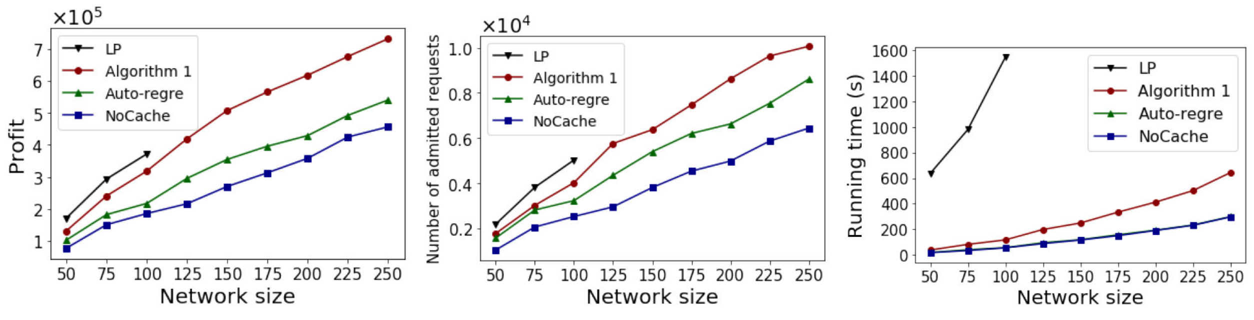
V. Performance Evaluation

In this section, we evaluate the performance of the proposed algorithm for the online service placement and request assignment problem. We also analyze the impact of important parameters on the performance of the proposed algorithm.

A. Experimental Settings

We consider an MEC network with 100 APs, where 20% of them are equipped with cloudlets. The topologies of the network is randomly generated through a tool GT-ITM [20]. The computing capacity of each cloudlet is value randomly selected from 2,000 MHz to 4,000 MHz [13]. We assume that the MEC network provides 20 different types of services, and each different type of service instance requires 40 to 400 MHz computing resources randomly [13]. The network delay on the link between two APs is set a random value from 2 *ms* to 5 *ms* [26]. The instantiation delay of a service instance varies from 20 *ms* to 40 *ms* [27]. The instantiation cost of a service instance is randomly drawn within [20, 50] [38]. The profit of processing a user request per time slot is set within [0.2, 0.3] per MHz [38], and the idle cost for a type of service instance is set within [0.02, 0.03] per MHz per time slot. The request arrivals of a certain type of services at an AP follows a Poisson distribution, where the mean of Poisson distribution λ is randomly chosen within [0.05, 0.1]. The delay requirements of requests vary from 10 *ms* to 50 *ms*, and each request lasts from 1 to 5 time slots [13]. The value in each figure is the mean of the results out of 50 network instances of the same size, and the running time is obtained based on a machine with a 3.79GHz AMD Ryzen 5 CPU, RTX 2070 GPU and 32GB RAM. Unless otherwise specified, these parameters will be adopted in the default setting. Table II summarizes the default settings of the parameters in this paper.

We evaluated Algorithm 1 against algorithm Auto-regre proposed in [38], where they utilized an auto-regression method to predict the number of required service instances as $n^t_{j,k} = 0.4 * n^{t-1}_{j,k} + 0.3 * n^{t-2}_{j,k} + 0.2 * n^{t-3}_{j,k} + 0.1 * n^{t-4}_{j,k}$. A baseline algorithm for service placement is also proposed, which releases idle service instances immediately when the demanded service by a request finishes, and it does not pre-install any service instances. We refer to this baseline algorithm as NoCache. For Auto-regression and NoCache, we propose a baseline algorithm for request assignment too, which assigns requests to cloudlets in descending order of their computing resource demands. For a given request r_i, we choose a set of cloudlets with sufficient computing capacities while meeting its service delay requirement. If there is not



(a) (b) (c)

Fig. 4. Performance of different algorithms by varying the size of the MEC network from 50 to 250.

TABLE II
TABLE OF PARAMETER SETTINGS

Parameters	Values
Number of APs	100
Number of cloudlets	20
Computing capacity	2,000 MHz - 4,000 MHz
Number of services	20
Required computing resource	40 MHz - 400 MHz
Network delay	2 ms - 5 ms
Instantiation delay	20 ms - 40 ms
Instantiation cost per MHz	20 - 50
Idle cost per MHz	20 - 50
Mean of arriving rate of requests per APs	0.05 - 0.1
Delay requirements	10 ms - 50 ms
Request duration	1 - 5 time slots

such a cloudlet, r_i will be rejected; otherwise, request r_i is assigned to the cloudlet with the maximum residual capacity and an idle instance for service f_{k_i}, i.e., r_i is assigned to a cloudlet c_j with the maximum cap_j by instantiating a new service instance on it. We also relax the ILP formulation to a linear programming (LP), and make use of the solution of the LP as an upper bound on the problem of concern. Even so, with the increase in network size (numbers of APs) or the number of requests, the running time of algorithm LP dramatically grows, the LP algorithm thus is only applicable when the problem size is small or moderate. We refer to the LP solution as `LP`.

B. Performance Evaluation of the Online Algorithm

We studied the performance of Algorithm 1 against algorithms `Auto-regression` and `NoCache`, respectively, by varying network size from 50 to 250. Fig. 4(a) plots the profits achieved by the three comparison algorithms and the upper bound on the optimal solution. It can be observed that Algorithm 1 delivers a solution which is from 76.4% to 80.5% of the upper bound of the optimal solution when network size varies from 50 to 100. Also, the profit by Algorithm 1 outperforms those delivered by algorithms `Auto-regression` and `NoCache` in all cases, and the performance gap between Algorithm 1 and the other comparison algorithms becomes larger with the increase on network size. When there are 250 APs, Algorithm 1 receives 35.4% more profits than that of algorithm `Auto-regression`. The rationale behind this is that Algorithm 1 has an advantage in the placement of service instances and delivers a better assignment of requests. Fig. 4(b) demonstrates the number of admitted requests by all algorithms. It can be seen that Algorithm 1 admits 24.5% more requests than that by algorithm `Auto-regression`, which shows that Algorithm 1 achieves better prediction of arriving requests.

Fig. 4(c) depicts the running times of the mentioned algorithms. Note that in reality, if the pattern of requests in the monitoring period does not dramatically change, we only need to conduct the training procedure once, as the trained DNNs can be re-used. Therefore, we exclude the training time from the running time of Algorithm 1. It can be seen that Algorithm 1 takes longer compared with algorithms `Auto-regression` and `NoCache`. Despite the running time of Algorithm 1 is high (637.12 seconds) when the network size is 250, the average running time per time slot is around 12.74 seconds, which is applicable since the proposed algorithm proceeds in an online manner. Also, Procedure 2 can be applied before the arrivals of requests, which means the running time on determining service placement does not affect the latency of requests. Moreover, Procedure 2 can be run in cloudlets distributively, which further reduces the running time. Therefore, Algorithm 1 is applicable for latency-sensitive services and large-scale environments.

C. Impact of Important Parameters on the Performance of the Online Algorithm

We now investigate the impact of important parameters on algorithms: The arriving rate of requests and delay requirements of requests on the performance of the online algorithm Algorithm 1, by varying the mean of the Poisson distribution λ from [0.025, 0.075] to [0.1, 0.15]. As shown in Fig. 5(a), Algorithm 1 always outperforms algorithms `Auto-regression` and `NoCache` under different arriving rates of requests. When the arriving rate of requests is set within [0.1, 0.15], the profit of Algorithm 1 is 37.30% more than that of algorithm `NoCache`. It also can be seen that the profit gap between Algorithm 1 and algorithm `NoCache` becomes smaller when λ is drawn in the interval [0.0625, 0.1125]. The rationale behind this is that with the increase on arriving rates of requests, algorithm `NoCache`

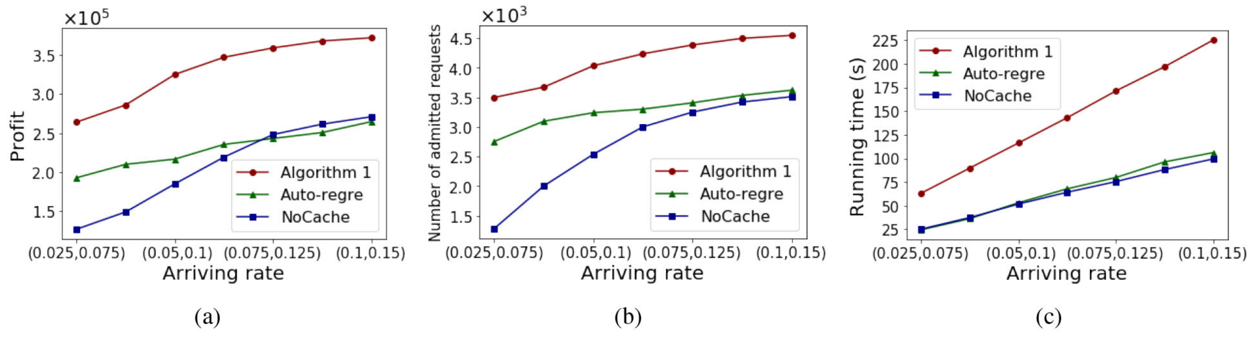


Fig. 5. Performance of different algorithms by varying the Possion distribution parameter λ from the interval [0.025, 0.075] to the interval [0.1, 0.15].

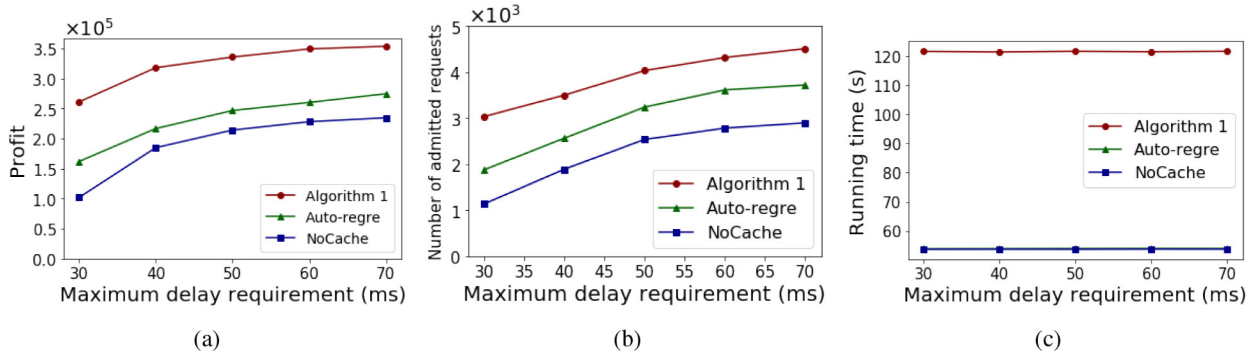


Fig. 6. Performance of different algorithms by varying the maximum delay requirement from 30*ms* to 70*ms*.

can accept more requests with flexible delay requirements, but Algorithm 1 cannot accept more requests due to the capacity constraint on each cloudlet. As shown in Fig. 5(b), Algorithm 1 admits the maximum number of user requests compared with the other algorithms. On the other hand, when the arriving rate of requests is in the interval [0.075, 0.125], algorithm `NoCache` admits similar numbers of requests as algorithm `Auto-regression`, but algorithm `Auto-regression` collects fewer profits than those of the other two algorithms. The reason is that algorithm `Auto-regression` does not consider the profit of requests, so it pre-installs service instances with fewer profits. Instead, algorithm `NoCache` has more computing resources for admitting requests with high profits. Meanwhile, Algorithm 1 learns to pre-install service instances with higher profits, which leads to a higher accumulative profit. In addition, Algorithm 1 finishes within an acceptable time in Fig. 5(c).

We finally study the impact of delay requirements of requests on the performance of different algorithms, by varying the delay from [10, 30] *ms* to [10, 70] *ms*. Fig. 6(a) shows the performance of different algorithms in different delay requirement settings. We can see that Algorithm 1 delivers a solution with the maximum profit among the comparison algorithms. With more stringent delay requirements imposed, all the three comparison algorithms will deliver fewer profits. However, the profit reduction of Algorithm 1 is slower, while algorithm `NOCache` is fast. The reason is that Algorithm 1 pre-installs some instances, which shortens the service latency dramatically, and therefore it shows more tolerance to stringent delay requirements. From Fig. 6(b), we can also see that the number of admitted requests increases with a longer maximum delay requirement. Fig. 6(c) depicts that the running times of the three comparison algorithms.

VI. Conclusion

In this paper, we studied the problem of online service placement and request assignment in an MEC network while meeting user service delay requirements. We proposed a deep reinforcement learning-based mechanism to predict the most demanded services and their placements in a foreseeable future. We also devised an efficient online algorithm for dynamic assigning service requests to cloudlets with and without existing service instances. We finally conducted experiments through simulations. Experimental results demonstrate that the proposed algorithm is efficient and superior to the benchmark algorithms.

References

[1] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz, "Reinforcement learning through asynchronous advantage actor-critic on a GPU," Nov. 18, 2016, *arXiv:1611.06256*.

[2] Y. S. Chow, H. Robbins, and D. Siegmund, "Great expectations: The theory of optimal stopping," *J. Roy. Stat. Soc. Series A*, vol. 135, no. 4, p. 610, 1972.

[3] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4005–4018, Jun. 2019.

[4] R. Cohen, L. Katzir, and D. Raz, "An efficient approximation for the generalized assignment problem," *Inf. Process. Lett.*, vol. 100, no. 4, pp. 162–166, 2006.

[5] T.H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, MA, USA: MIT Press, 2009.

[6] R. Cziva, C. Anagnostopoulos, and D. P. Pezaros, "Dynamic, latency-optimal VNF placement at the network edge," in *Proc. INFOCOM*, 2018, pp. 693–701.

[7] J. Du, L. Zhao, J. Feng, X. Chu, and F. R. Yu, "Economical revenue maximization in cache enhanced mobile edge computing," in *Proc. ICC*, 2018, pp. 1–6.

[8] H. O. Hassan, S. Azizi, and M. Shojafar, "Priority, network and energy-aware placement of IoT-based application services in fog-cloud environments," *IET commun.*, vol. 14, no. 13, pp. 2117–2129, 2020.

[9] T. He, H. Khamfroush, S. Wang, T. La Porta, and S. Stein, "It's hard to share: Joint service placement and request scheduling in edge clouds with sharable and non-sharable resources," in *Proc. ICDCS*, 2018, pp. 365–375.

[10] Y. He, F. R. Yu, N. Zhao, V. C. M. Leung, and H. Yin, "Software-defined networks with mobile edge computing and caching for smart cities: A big data deep reinforcement learning approach," *IEEE Commun. Mag.*, vol. 55, no. 12, pp. 31–37, Dec. 2017.

[11] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol.9, no.8, pp. 1735–1780, 1997.

[12] L. Huang, S. Bi, and Y.-J. A. Zhang, "Deep reinforcement learning for Online computation offloading in wireless powered mobile-edge computing networks," *IEEE Trans. Mobile Comput.*, vol. 19, no. 11, pp. 2581–2593, Nov. 2020.

[13] M. Jia, W. Liang, and Z. Xu, "QoS-aware task offloading in distributed cloudlets with virtual network function services," in *Proc. MSWiM*, 2017, pp. 109–116.

[14] P. Kayal and J. Liebeherr, "Distributed service placement in fog computing: An iterative combinatorial auction approach," in *Proc. ICDCS*, 2020, pp. 2145–2156.

[15] J. Kiefer and J. Wolfowitz, "Stochastic estimation of the maximum of a regression function," *Ann. Math. Stat.*, vol. 23, no. 3, pp. 462–466, 1952.

[16] Y. C. Lee, C. Wang, A. Y. Zomaya, and B. B. Zhou, "Profit-driven service request scheduling in clouds," in *Proc. CCGrid*, 2010, pp. 15–24.

[17] J. Li, W. Liang, M. Huang, and X. Jia, "Reliability-aware network service provisioning in mobile edge-cloud networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 7, pp. 1545–1558, Jul. 2020.

[18] J. Li, W. Liang, Y. Li, Z. Xu, and X. Jia, "Delay-aware DNN inference throughput maximization in edge computing via jointly exploring partitioning and parallelism," in *Proc. LCN*, 2021, pp. 193–200.

[19] J. Li, W. Liang, Y. Li, Z. Xu, X. Jia, and S. Guo, "Throughput maximization of delay-aware DNN inference in edge computing by exploring DNN model partitioning and inference parallelism," *IEEE Trans. Mobile Comput.*, early access, Nov. 13, 2021, doi: 10.1109/TMC.2021.3125949.

[20] J. Li, W. Liang, and Y. Ma, "Robust service provisioning with service function chain requirements in mobile edge computing," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 2, pp. 2138–2153, Jun. 2021.

[21] J. Li *et al.*, "Maximizing user service satisfaction for delay-sensitive IoT applications in edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 5, pp. 1199–1212, May 2022.

[22] Y. Li, W. Liang, and J. Li, "Profit maximization for service placement and request assignment in edge computing via deep reinforcement learning," in *Proc. 24th Int. Conf. Model. Anal. Simulat. Wireless Mobile Syst.*, 2021, pp. 51–55.

[23] J. Liu, W. Lu, F. Zhou, P. Lu, and Z. Zhu, "On dynamic service function chain deployment and readjustment," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 3, pp. 543–553, Sep. 2017.

[24] N. Liu *et al.*, "A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning," in *Proc. ICDCS*, 2017, pp. 372–382.

[25] Y. Liu, H. Yu, S. Xie, and Y. Zhang, "Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 11, pp. 11158–11168, Nov. 2019.

[26] Y. Ma, W. Liang, Z. Xu, and S. Guo, "Profit maximization for admitting requests with network function services in distributed clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 5, pp. 1143–1157, May 2019.

[27] J. Martins *et al.*, "ClickOS and the art of network function virtualization," in *Proc. NSDI*, 2014, pp. 459–473.

[28] M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu, and W. Zhuang, "Learning-based computation offloading for IoT devices with energy harvesting," *IEEE Trans. Veh. Technol.*, vol. 68, no. 2, pp. 1930–1941, Feb. 2019.

[29] Z. Ning *et al.*, "Distributed and dynamic service placement in pervasive edge computing networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 6, pp. 1277–1292, Jun. 2021.

[30] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Joint service placement and request routing in multi-cell mobile edge computing networks," in *Proc. INFOCOM*, 2019, pp. 10–18.

[31] A. Samanta and Z. Chang, "Adaptive service offloading for revenue maximization in mobile edge computing with delay-constraint," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 3864–3872, Apr. 2019.

[32] Q. Sun, P. Lu, W. Lu, and Z. Zhu, "Forecast-assisted NFV service chain deployment based on affiliation-aware VNF placement," in *Proc. GLOBECOM*, 2016, pp. 1–6.

[33] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 1057–1063.

[34] M. Tang and V. W. S. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Trans. Mobile Comput.*, early access, Nov. 10, 2020, doi: 10.1109/TMC.2020.3036871.

[35] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung, "Dynamic service placement for mobile micro-clouds with predicted future costs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 1002–1016, Apr. 2017.

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

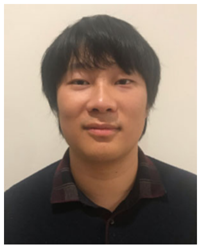
[37] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *Proc. INFOCOM*, 2018, pp. 207–215.

[38] Z. Xu, W. Liang, M. Jia, M. Huang, and G. Mao, "Task offloading with network function requirements in a mobile edge-cloud network," *IEEE Trans. Mobile Comput.*, vol. 18, no. 11, pp. 2672–2685, Nov. 2019.

[39] Z. Xu, Z. Zhang, W. Liang, Q. Xia, O. Rana, and G. Wu, "QoS-aware VNF placement and service chaining for IoT applications in multi-tier mobile edge networks," *ACM Trans. Sensor Netw.*, vol. 16, no. 3, pp. 1–27, 2020.

[40] Z. Yang, Y. Cui, X. Wang, Y. Liu, M. Li, and Z. Zhang, "Towards maximal service profit in geo-distributed clouds," in *Proc. ICDCS*, 2019, pp. 442–452.

[41] A. Yousefpour *et al.*, "FOGPLAN: A lightweight QoS-aware dynamic fog service provisioning framework," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 5080–5096, Jun. 2019.



Yuchen Li received the B.Sc. degree (First Class Hons.) in computer science from the Australian National University, in 2018. He is currently pursuing the Ph.D. degree with the Research School of Computing, The Australian National University. His research interests include the Internet of Things, mobile edge computing, and algorithm design.

Weifa Liang (Senior Member, IEEE) received the B.Sc. degree in computer science from Wuhan University, China in 1984, the M.E. degree in computer science from the University of Science and Technology of China in 1989, and the Ph.D. degree in computer science from the Australian National University in 1998. He is a Professor with the Department of Computer Science, City University of Hong Kong. Prior to that, he was a Professor with 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, network function virtualization, Internet of Things, software-defined networking, design and analysis of parallel and distributed algorithms, approximation algorithms, combinatorial optimization, and graph theory. He currently serves as an Associate Editor in the Editorial Board of IEEE TRANSACTIONS ON COMMUNICATIONS.

Jing Li received the B.Sc. degree (First Class Hons.) in computer science from the Australian National University in 2018. He is currently pursuing the Ph.D. degree with the Research School of Computing, The Australian National University. His research interests include mobile edge computing, network function virtualization, and combinatorial optimization.