This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2026.3747852



# QoE-Aware Model-Driven Inference Service Provisioning in Edge-Cloud Networks

Yuncan Zhang, *Member, IEEE*, Weifa Liang, *Fellow, IEEE*, Yizhun Zhang, and Yuanyuan Yang, *Life Fellow, IEEE*

***Abstract*—With the explosive growth of mobile devices and development of Artificial Intelligence (AI) applications, we are witnessing unprecedented growth of intelligent services based on machine learning models. The convergence of AI and edge computing has catalyzed growing interest in edge intelligence to enable model-driven inference services in edge devices. In this paper, we investigate a novel Quality of Experience (QoE)-aware service satisfaction maximization problem in an edge-cloud network, with the aim to maximize the accumulative service satisfaction and payoff of user requests while minimizing the total cost of resources consumed on services over a given time horizon. To this end, we first formulate a Nonlinear Integer Programming (NLIP) solution to the offline version of the problem. Then, we cast the problem as a Markov Decision Process (MDP), and devise a Deep Reinforcement Learning (DRL) algorithm for the MDP. The proposed DRL algorithm consists of a Deep Q-Network (DQN) to decide the number of pre-deployed configuration instances for each requested service model, followed by allocating sub-channels to requests and then assigning sub-channel-chosen requests to existing or newly instantiated model instances at each time slot. Finally, we evaluate the performance of the proposed algorithm via simulations. Simulation results demonstrate that the proposed algorithm is promising.**

***Index Terms*—Edge-cloud computing; model configuration instances; DRL algorithm; differential accuracy model instances; QoE; request assignments; inference model updating.**

## I. Introduction

The rapid proliferation of mobile devices and AI-driven applications has led to an exponential rise in intelligent services powered by machine learning models, such as Deep Neural Networks (DNNs) [41]. Given the substantial computing and storage demands inherent in intelligent services for data-intensive computing and analysis, clouds naturally serve as ideal platforms due to abundant resources. However, providing delay-sensitive intelligent services in clouds is inappropriate due to real-time response requirements, such as video analytics and health/event monitoring. For these applications, responding to an inference service request in a remote cloud will introduce a non-negligible backhaul latency, and raise privacy and security concerns, thereby not meeting stringent service delay requirements of users. Fortunately, edge computing by deploying computing and storage resources at the edge of core networks is capable of addressing the mentioned limitations, enabling AI and deep learning technology for data processing and decision-making in cloudlets and edge devices [3], [34], [35], [37], [38], [42].

To provide delay-sensitive yet accurate inference services in edge computing, we focus on drift-sensitive inference services whose performance may degrade as data distributions evolve over time, such as health monitoring, video analytics, and digital-twin-assisted sensing [14], [20], [39]. For such services, their inference models need to be refreshed over time using newly generated source data. However, a resource-constrained edge computing network is unable to accommodate and train large numbers of service models. Consequently, hosting millions of service models and continual training on the models usually are performed in the remote cloud, and service models in the cloud are referred to as *primary service models*, which are assumed to be the most up-to-date ones. The primary service models in the cloud can be cached to edge servers in a Mobile Edge Computing (MEC) network, where the cached model instances are periodically or on-demand replaced by the latest primary models from the cloud to ensure their service accuracy to end users [22].

User satisfaction with a service usually is determined by two critical factors: (i) whether the requested service is accurate; and (ii) whether the service request is responded on time. To ensure service accuracy, service models need to be retrained with the updated data from their source devices [40]. To maintain the freshness of source data of a service model, the Age of Information (AoI) has been widely adopted to measure the freshness of information [14], [32], as it can dynamically monitor the sources of information and the way the information is used. Following prior studies [16], [20], [21], we here adopt the AoI metric to measure the freshness of a service model instance. On the other hand, initializing a new model instance usually incurs much higher latency compared to performing inferences on an existing model instance. The model instantiation delay may cause the end-to-end service delay of an incoming request to exceed its delay requirement, resulting in request rejection. To mitigate instantiation delays of service model instances, one efficient approach is to deploy most demanded model instances in advance, and assign incoming requests to them directly. Furthermore, to fully utilize limited resources in MEC while meeting diverse QoE requirements of users, service models can support different configuration instances with different amounts of computing and memory resource allocated, where a model instance with more compute/memory resource demands usually has a lower service delay [2]. The different model configuration instances enable adaptive resource provisioning tailored to different user service demands. In this paper, we consider QoE-aware

Corresponding author: Weifa Liang (email: weifa.liang@cityu.edu.hk)

Y. Zhang is with the School of Software Engineering, Sun Yat-Sen University, Zhuhai, P. R. China. Email: zhangyc29@mail.sysu.edu.cn

W. Liang and Y. Zhang are with the Department of Computer Science, City University of Hong Kong, Hong Kong, P. R. China. Emails: weifa.liang@cityu.edu.hk, and yzhang5624-c@my.cityu.edu.hk

Y. Yang is with the Department of Electrical and Computer Engineering, Stony Brook University, Stony Brook, NY, USA. Email: yuanyuan.yang@stonybrook.edu

Authorized licensed use limited to: CITY UNIV OF HONG KONG. Downloaded on November 01,2026 at 03:34:31 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2026.3747852



inference services in an edge-cloud network, with the aim to maximize the cumulative service satisfaction and payoff of users while minimizing the total cost of resources consumed on the service provisioning. This poses the following challenges.

First, each request has a stringent delay requirement, which requires its requested model instance to be placed at a location not far from its user location. Also, choosing an appropriate model instance while meeting the delay requirement of a request admission is challenging, as there are up to $K$ different configuration instances to choose. Thus choosing which model configuration instance and placing the instance to which cloudlet is critical. Second, the instantiation of a model instance usually takes several orders of magnitude longer than the inference time on the model instance. To reduce instantiation delays of model instances, it is a common practice to deploy model instances in advance. However, the pre-deployment of model instances consumes both computing and memory resources. Then, how to determine the number of future requests for different model services and to determine how many model configuration instances for each service model is challenging, too. Third, as the freshness of a service model determines its service accuracy, how often should a service model be updated by downloading its updated version from a remote cloud? and when should a service model be downloaded? These decisions will affect service accuracy of service models, user satisfaction on services and service costs. Finally, user mobility and changing service request patterns heavily impact their requested model instance placements. Accurately determining different numbers of instances of service models demanded for further requests is critical, as this can help mitigate model instance instantiation delays, thereby admitting more requests. This paper will address the aforementioned challenges.

The novelty of this work lies in the study of the QoE-aware service satisfaction maximization problem in an edge-cloud network, by jointly optimizing model management and QoE-aware service provisioning in collaborative edge-cloud environments. To tackle the problem efficiently, a two-layer online optimization framework is proposed that decouples this long-term optimization problem into a series of one-time-slot optimization problems, which involve different model caching and replacements, and assigning requests to their inference models in different cloudlets, subject to limited available bandwidth and computing resources.

The main contributions of this paper are summarized as follows. We first formulate a novel QoE-aware service satisfaction maximization problem within a given time horizon, where a set of service requests arrive in the beginning of each time slot, and each incoming request is either admitted or rejected at the time slot. We propose a Nonlinear Integer Program (NLIP) solution to the offline version of the problem. We then develop a two-layer online optimization framework for the problem, in which a DQN module learns model instance pre-deployments and model updating decisions at the top layer, and efficient algorithms are designed to admit incoming requests at the bottom layer. Specifically, at each time slot, we deal with assigning sub-channels to incoming requests and allocating the sub-channel-chosen requests to cloudlets that host their requested model instances. We implement these two subtasks by adopting the maximum matching technique and reducing to the maximum profit general assignment problem, respectively. Finally, we conduct simulations to evaluate the performance of the proposed algorithm. Simulation results demonstrate that the proposed algorithm is promising.

The rest of the paper is organized as follows. Section II reviews related works on this topic. Section III introduces the system model, notions, notations, and problem definition. Section IV formulates an NLIP solution for the offline version of the problem. Section V devises an online algorithm for the problem. Section VI evaluates the performance of the proposed algorithm, and Section VII concludes the paper.

## II. Related work

**Service satisfaction.** User satisfaction on services provided by MEC networks has been extensively investigated in past several years [17], [18], [19]. For example, Li *et al.* [19] dealt with the QoE-aware service satisfaction maximization problem of dynamic Service Function Chain (SFC)-enabled request admissions, and devised a DRL algorithm for the online service instance placement and request assignment problem. Li *et al.* [17] considered a budget-aware user satisfaction maximization of SFC-enabled services, where the satisfaction of a user on his requested service depends on the service reliability. They developed efficient algorithms with provable approximation ratio for the problem.

**AoI-aware DT-assisted model freshness.** The AoI of a service model and Digital Twin (DT)-assisted model freshness have attracted considerable attention recently [1], [13], [14], [20], [31], [39]. For instance, Li *et al.* [14] devised approximation and online algorithms for user satisfaction on services through dynamic DT placements to provide fresh AoI DT services. Li *et al.* [15] and Zhang *et al.* [39] considered AoI-aware inference services in an MEC through network slicing, and developed efficient approximation and online algorithms for DT network slicing. Li *et al.* [16] studied a joint freshness optimization of DT statuses and user service queries on DT data in an MEC. Yang *et al.* [31] studied DT-assisted task executions by presenting a two-timescale accuracy-aware online algorithm, to maximize the accuracy of each task execution while meeting both energy and delay constraints. Li *et al.* [20] studied AoI-aware fidelity query services in a DT-assisted MEC with a given budget, by proposing an optimization and evaluation framework.

**Service accuracy and service delay.** To strive a fine trades-off between service accuracy and inference delay, multiple resolution inference service provisions [37], [41] and resource allocation of service model instances [5], [33] in MECs were also investigated. Zhao *et al.* [41] developed an online optimization framework `EdgeAdaptor` for DNN inference services in either a single edge server or a remote cloud. They formulated a cost minimization problem that jointly optimizes the application configuration adaption, DNN model selection, and resource provisioning. Zhang *et al.* [37] considered the profit maximization problem of delay-sensitive,

Authorized licensed use limited to: CITY UNIV OF HONG KONG. Downloaded on November 01,2026 at 03:34:31 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2026.3747852



TABLE I
COMPARISON OF PROBLEM SCOPE AND SOLUTION FEATURES WITH RELATED WORKS

| Feature\Work | [11] | [14] | [16] | [20] | [22] | [26] | [31] | Ours |
|---|---|---|---|---|---|---|---|---|
| Model/service/DT placement | × | ✓ | ✓ | × | ✓ | ✓ | ✓ | ✓ |
| Service/model caching | × | × | × | × | ✓ | ✓ | × | ✓ |
| Model/data freshness awareness | × | ✓ | ✓ | ✓ | ✓ | × | ✓ | ✓ |
| Heterogeneous model configurations | × | × | × | × | × | × | × | ✓ |
| Batched inference capacity | × | × | × | × | × | × | × | ✓ |
| Joint model management and request provisioning | × | × | × | × | ✓ | × | ✓ | ✓ |
| DRL-based algorithm | ✓ | × | × | × | × | ✓ | × | ✓ |

differential accuracy inference services in MEC. They developed performance-guaranteed algorithms for different model instance choices, DT placements, and request assignment to model instances. Zhang *et al.* [33] formulated a joint inference task assignment and DNN model placement problem in an end-edge-cloud architecture to maximize the average inference accuracy while meeting task delay requirements, and developed a DRL algorithm for the problem. Chen *et al.* [5] dealt with the DNN model caching and processor allocation problem while meeting user delay and energy consumption constraints. They proposed a near optimal solution with high probability, by leveraging the Lyapunov optimization technique.

**Service model deployment and request assignment.** A few recent studies investigated on-demand service model caching in MEC, where service models that are continually trained in remote clouds need to be refreshed locally by downloading from the remote cloud. For instance, Liu *et al.* [22] studied a joint optimization problem of service model deployment and task assignment in an MEC, with the aim to maximize service QoE and payoffs while minimizing the cost of various resource consumed. They decomposed this joint optimization problem into two sub-problems: service model deployments and request assignments, and designed algorithms for them separately. Gao *et al.* [11] considered deploying DNN models for video analytics at edge nodes by jointly optimizing the recognition accuracy and the overall delay. Xu *et al.* [28] developed a reinforcement learning algorithm for collaborative DNN model selection in heterogeneous edge computing systems, with the aim of improving inference accuracy while reducing the system overhead. Xia *et al.* [26] considered personalized computation offloading and service caching in MEC with heterogeneous networks, and proposed a DRL algorithm to minimize both task processing delay and the energy consumption in mobile devices.

Unlike the aforementioned studies, in this paper, we study the QoE-aware service satisfaction maximization problem in an edge-cloud network, with the aim to maximize the accumulative service satisfaction and the total payoff of users while minimizing the total cost of various resources consumed on services simultaneously. We jointly consider dynamic placements of different model instances of each service model, dynamic model updating for model freshness, and assignments of service requests to model instances while meeting their delay requirements. Table I highlights the differences of our work from the aforementioned existing studies.

## III. PRELIMINARIES

In this section, we first introduce the system model, notions and notations. We then define the problems precisely.

### A. System model

We consider an edge-cloud computing environment. There is a remote cloud $c_0$ with abundant resources, which accommodates a set $\mathcal{M}$ of models for inference services. We assume that there is an edge-cloud MEC network $G = (N \cup \{c_0\}, E)$ that consists of a set $N$ of Access Points (APs) and a set $E$ of links between APs. Associated with each AP, there is a co-located cloudlet and the communication delay between the AP and its co-located cloudlet is negligible. Unless otherwise specified, we use an AP and its co-located cloudlet interchangeably if no confusion arises. Denote by $C_j$ the computing capacity on cloudlet $j \in N$, and denote by $d_e$ and $\zeta_e$ the transmission delay and the cost per unit data transfer along link $e \in E$, respectively. All cloudlets can communicate with the remote cloud $c_0$ through a gateway node, and the transmission delay is larger than the transmission delay between any two cloudlets.

To deliver high-quality services to end users, user-demanded service models in the cloud are cached to cloudlets in the MEC to enable efficient service provisioning to their users. Also, there are multiple requests for the same service model, each with differentiated delay requirements. Thus, given a service model, it may have different configuration instances in the MEC with different amounts of computing and storage resources allocated for each of them. For example, consider two different configuration instances of a service model $M_m$: one with 2vCPU and 2GiB memory; and another with 4vCPU and 8GiB memory, respectively. Usually the amount of storage resource allocated to a model configuration instance is proportional to the amount of computing resource it demands. Assume that the number of different model configuration instances of each service model $M_m \in \mathcal{M}$ is no more than $K \geq 1$, which are represented as model instances $M_m^{(1)}, M_m^{(2)}, \ldots, M_m^{(K)}$, and the amounts of computing resources allocated to these different model instances are $comp_m^{(1)}, comp_m^{(2)}, \ldots, comp_m^{(K)}$, respectively, where $K$ is a small constant and $comp_m^{(1)} < comp_m^{(2)} < \ldots < comp_m^{(K)}$. Given a request for model $M_m$, if the request is processed by a model instance $M_m^{(k)}$ or $M_m^{(k')}$ with $k < k'$, then its inference delay on instance $M_m^{(k')}$ is less than that on instance $M_m^{(k)}$ due to that more computing and memory resources are allocated to instance $M_m^{(k')}$. In addition, we assume that each model instance is capable of processing multiple requests simultaneously at each time slot. Let $L_{m,k}$ denote the batch-processing capacity of one instance of $M_m^{(k)}$, i.e., the maximum number of requests it can process within one time slot. In practical edge AI systems, a cloudlet may be equipped with different types of computing resources such as CPUs, GPUs, NPUs, or other AI accelerators. We interpret computing capacity $C_j$ on cloudlet $j \in N$ and the resource demand $comp_m^{(k)}$ of model instance $M_m^{(k)}$ as the accelerator-normalized compute capacity rather than just the CPU frequency of cloudlet $j$ only. They represent the total amount of compute

Authorized licensed use limited to: CITY UNIV OF HONG KONG. Downloaded on November 01,2026 at 03:34:31 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2026.3747852



and memory resources reserved for hosting model instances, after accounting for the serving capability of the underlying hardware. Thus, different configuration instances $M_m^{(k)}$ capture different levels of accelerator and memory reservations.

We adopt the AoI to measure the freshness of a cached model instance in a cloudlet, recall that the primary model of each service model is always kept in the remote cloud. We thus assume that a downloaded primary model from the cloud at the current time slot is the freshest one, and the freshness of a service model indicates that the cached model instance is more conducive to improving its inference accuracy. We further assume that the AoIs of all different instances of each service model in the MEC are identical, implying whenever there is a downloading of its primary service model from the remote cloud, all different configuration instances of the model in the MEC will be updated accordingly, and their AoIs become 1. Such updating can be implemented by a multicasting operation. The AoI of each model instance in the MEC linearly increases with the number of time slots passed since its last downloading from the remote cloud [14], [22].

Given a time horizon $\mathbb{T}$, a set $U_t$ of requests arrives in the beginning of each time slot $t \in \mathbb{T}$. Each arrived request must be responded by either an acceptance or a rejection in the end of the time slot. Each request $r_i \in U_t$ is represented by a tuple $\langle \mathcal{C}_i, vol_i, m_i, D_i \rangle$, where $\mathcal{C}_i \subset N$ is the set of APs within the maximum transmission range of the user of request $r_i$, $vol_i$ is the volume of the input of request $r_i$, $m_i$ is the index of service model requested by $r_i$ with $M_{m_i} \in \mathcal{M}$, and $D_i$ is the service delay requirement of request $r_i$ that consists of (1) the instantiation delay of the requested model instance if it has not been instantiated yet, (2) the data uploading delay, (3) the data transfer delay, and (4) the request processing delay on its requested model instance, which is hosted by a Virtual Machine (VM) or a container in a cloudlet [27]. Fig. 1 is an illustrative example of the system model that consists of the remote cloud and an MEC network. The cloud hosts all service models and maintains the freshness of these models through continual training, using updated source data. The MEC network consists of five APs and co-located cloudlets with optical links between APs, and mobile users issue their service requests to the MEC for inference services.

### B. Service delays of service requests

To successfully admit a service request, there must be a model instance of its requested model in a cloudlet to process the request while meeting the delay requirement of the request. The requested model instance can be either pre-deployed in a cloudlet and the request can be assigned to the instance immediately, or a new instance can be instantiated for the request with the extra instantiation delay. The end-to-end delay of a request consists of the uploading delay of the input, the transmission delay from its current location to the cloudlet hosting its requested model instance, and the processing delay on the model instance. We assume that the amount of bandwidth allocated to each request under an AP is equal, by adopting the Orthogonal Frequency-Division Multiple Access (OFDMA) scheme. Let $R_{i,l_i}$ be the uploading

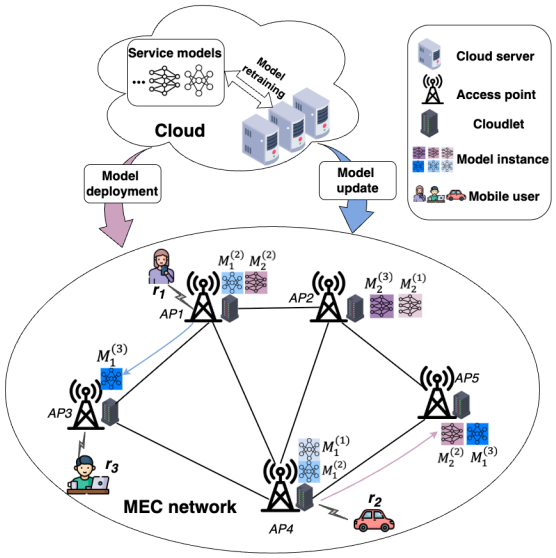



Fig. 1. An illustrative example of an edge-cloud network with two service models $M_1$ and $M_2$ that support three distinct configuration instances, where different configuration instances of model $M_1$ are $M_1^{(1)}$, $M_1^{(2)}$, and $M_1^{(3)}$. Service request $r_1$ that requests for model $M_1$ with a low delay requirement is assigned to the model instance $M_1^{(3)}$ in cloudlet 3, and service request $r_2$ requesting model $M_2$ with a medium delay requirement is assigned to the model instance $M_2^{(2)}$ in cloudlet 5.

rate of request $r_i$ via AP $l_i \in N$ with bandwidth $B_{l_i}$ and $l_i \in \mathcal{C}_i$, and then we have $R_{i,l_i} = \frac{B_{l_i}}{L_{l_i}} \cdot \log(1 + \frac{PX_i \cdot H_{i,l_i}}{\sigma^2 + I_{l_i}})$, where $B_{l_i}/L_{l_i}$ is the amount of bandwidth of AP $l_i$ allocated to request $r_i$, $PX_i$ is the transmission power of the user device, $H_{i,l_i}$ is the channel gain between the location of request $r_i$ and AP $l_i$, $L_{l_i}$ is the number of subchannels on AP $l_i$, and $\sigma^2 + I_{l_i}$ is the average power of the noise and interference over the bandwidth [30]. Thus, the uploading delay of request $r_i$ is $vol_i/R_{i,l_i}$. Further, assume that request $r_i$ is assigned to model instance $M_{m_i}^{(k)}$ in cloudlet $j$ at time slot $t$. Let $path_{l_i,j}$ be a shortest path in $G$ between $AP_{l_i}$ and $AP_j$. The transmission delay per unit data along path $path_{l_i,j}$ is $\sum_{e \in path_{l_i,j}} d_e$. The end-to-end delay $d_{i,j,l_i,m_i,k}$ of request $r_i$ thus is

$$d_{i,j,l_i,m_i,k} = \frac{vol_i}{R_{i,l_i}} + vol_i \cdot \sum_{e \in path_{l_i,j}} d_e + \tau_{m_i,k}, \quad (1)$$

where $\tau_{m_i,k}$ is the processing delay of request $r_i$ by instance $M_{m_i}^{(k)}$ in cloudlet $j$.

To meet the delay requirement $D_i$ of request $r_i$, $d_{i,j,m_i,k} + init_{m_i,k} \leq D_i$ if it is assigned to a newly instantiated instance $M_{m_i}^{(k)}$, where $init_{m_i,k}$ is used to model the instantiation latency on launching instance $M_{m_i,k}$, including model loading into memory or accelerator memory, container/VM initialization, accelerator resource allocation, and runtime warm-up, which motivates pre-deploying model instances to further reduce request admission delay; otherwise $d_{i,j,m_i,k} \leq D_i$.

### C. Service payment, resource consumption cost, and user satisfaction

We consider three components in the admission of a request: its service payment, its resource consumption, and its user satisfaction on the service.

Authorized licensed use limited to: CITY UNIV OF HONG KONG. Downloaded on November 01,2026 at 03:34:31 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2026.3747852



**The monetary payment of a request admission.** Network service providers typically adopt a pay-as-you-go billing scheme [23]. We here adopt the payment scheme of an admitted request $r_i$ as a stair-step function $\psi(\cdot,\cdot)$ as follows [37].

$$pay_i = \psi(m_i, D_i), \tag{2}$$

where $m_i$ is the index of its requested service model $M_{m_i}$, $D_i$ is its service delay requirement, and $\psi(m_i, D_i)$ is a monetary function of $M_{m_i}$ and $D_i$. The range of delay $D_i$ can be divided into several intervals, and a request with a larger delay requirement results in a lower payment compared with another request for the same service but with a smaller delay requirement, i.e., $pay_i < pay_{i'}$ if they request for the same model service $M_{m_i} = M_{m_{i'}}$ but $D_i > D_{i'}$. The service provider can adjust the payment function according to the market demands.

**The updating cost of a service model.** We adopt the AoI metric to measure the freshness of a service model downloaded from the remote cloud. If the latest version of service model $M_m$ is made available in the MEC at the beginning of time slot $t$, then its AoI is reset to 1, i.e., $AoI(M_m, t) = 1$; otherwise $AoI(M_m, t) = AoI(M_m, t-1) + 1$ if there is no updating on model $M_m$ at time slot $t$. Once model $M_m$ is selected for updating, the MEC downloads the primary service model from the remote cloud. The cloudlet receiving the model then multicasts it to all remaining instances of $M_m$ in the MEC network via a multicast tree rooted at that cloudlet and spanning all cloudlets hosting the model instances.

Let $Update_cost(M_m, t)$ be the updating cost of model instances of model $M_m$ at time slot $t$. Then,

$$Update_cost(M_m, t) = \eta \cdot \mu_m + Multi_cast(M_m, t), \tag{3}$$

where $\eta$ is the cost of downloading a unit data of model $M_m$ from the cloud, $\mu_m$ is the number of model parameters of $M_m$, and $Multi_cast(M_m, t)$ is the updating cost of all instances of $M_m$ in the MEC at time slot $t$, which is defined as follows. Assuming that cloudlet $j$ is the receiver of model $M_m$ downloaded from the remote cloud, let $\mathcal{N}(M_m, t)$ be the set of cloudlets hosting at least one instance of $M_m$. A multicast tree $T(M_m, t)$ in $G$ is constructed, which is a tree rooted at node $j$ and spanning all nodes in $\mathcal{N}(M_m, t)$. An approximate solution for finding the multicast tree is twice the optimal one [12]. Then,

$$Multi_cast(M_m, t) = \mu_m \cdot \sum_{e \in T(M_m, t)} \zeta_e, \tag{4}$$

where $\zeta_e$ is the cost of unit data transfer along link $e \in E$, and $\mu_m$ is the number of parameters of model $M_m$.

**Request admission cost.** A request admission is to assign it to its requested model instance in a cloudlet, and the end-to-end response delay is no greater than its delay requirement. Denote by $\varphi_{j,m_i,k}$ the cost of processing request $r_i$ on a model instance $M_{m_i}^{(k)}$ in cloudlet $j$, which is a given constant [37]. The cost of admitting request $r_i$ thus is

$$cost_{i,j,l_i,m_i,k} = \xi \cdot \frac{vol_i}{R_{i,l_i}} + vol_i \cdot \sum_{e \in path_{l_i,j}} \zeta_e + \varphi_{j,m_i,k}, \tag{5}$$

where the first term in the RHS of Eq. (5) is the cost of uploading of request $r_i$, the second term is the data transfer cost from its user current location to the cloudlet hosting $M_{m_i}^{(k)}$, and the last term is the processing cost of request $r_i$ on the model instance. $\xi$ is the uploading cost per time unit, and $\zeta_e$ is the cost per unit data transfer on link $e \in E$.

**User satisfaction on services.** The QoE experienced by a user on a service usually is closely related to service fidelity of his requested model. Due to user mobility, environmental dynamics, and changing sensing conditions, the source data provided for model training evolves over time, an inference model may become stale if it is not retrained using the updated source data for a while. This implies that a cached model with a large AoI indicates that there is a big freshness gap between the cached model and its primary version in the cloud, and the model staleness may degrade its service fidelity. Therefore, a fresh service model with low AoI generally delivers high service fidelity. To quantify user satisfaction on different model services, we introduce a monotonically decreasing function $f(M_m, t)$ with respect to AoI to capture the freshness of service model $M_m$ at time slot $t$. In particular, we adopt a submodular function $f(\cdot)$ as a representative example to model the diminishing returns of model freshness, i.e., $f(M_m, t) = a^{\frac{1}{AoI(M_m,t)}}$, where $AoI(M_m, t)$ is the AoI of a model instance of $M_m$ at time slot $t$, and $a > 1$ is constant. It can be seen that a larger $a$ implies user satisfaction is more sensitive to the model freshness, and a small degradation of model freshness may impact on user satisfaction significantly. Otherwise, a small $a$ indicates that user satisfaction is less sensitive to model freshness.

The user satisfaction $q(r_i, t)$ of request $r_i$ at time slot $t$ is determined by the freshness of its requested service model $M_{m_i}$, which is defined as

$$q(r_i, t) = f(M_{m_i}, t) = a^{\frac{1}{AoI(M_{m_i},t)}}. \tag{6}$$

For request $r_i$, its user satisfaction is measured by a value of an exponential function with base $a > 1$ when its requested model $M_{m_i}$ is downloaded from the remote cloud at time slot $t$; otherwise its user satisfaction is diminishing with time duration since the last downloading of model $M_{m_i}$.

### *D. Problem definition*

We study a QoE-aware service satisfaction maximization problem, with the aim to maximize the linear combination of the long-term user satisfaction and payoff of admitted requests while minimizing the total cost of various resources consumed on service provisioning. Specifically, given a time slot $t$, let $x_{i,j,m,k}^t$ and $y_{i,j,m,k}^t$ be binary variables that indicate whether request $r_i \in U_t$ is assigned to a newly instantiated or an existing model instance $M_m^{(k)}$ in cloudlet $j \in N$ at time slot $t$, and let $n_{j,m,k}^t$ and $w_{j,m,k}^t$ denote the numbers of pre-deployed and newly instantiated model instances of $M_m^{(k)}$ in cloudlet $j$ at time slot $t$, respectively. Let $z_{m,t}$ be a binary variable indicating whether all instances of model $M_m$ in the MEC need to update at time slot $t$. Let $g_{i,l_i}^t$ be a binary variable that indicates whether the device of request $r_i$ uploads its input via

Authorized licensed use limited to: CITY UNIV OF HONG KONG. Downloaded on November 01,2026 at 03:34:31 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2026.3747852



$AP_{l_i}$ at time slot $t$. The optimization objective at time slot $t$ is to maximize

$$\begin{aligned}
&OBJ(U_t, N, \mathcal{M}, X_t, Y_t, t)\\
&= \sum_{r_i \in U_t} \sum_{j \in N} \sum_{k \in [K]} \alpha \cdot q(r_i, t) \cdot (x^t_{i,j,m_i,k} + y^t_{i,j,m_i,k})\\
&+ \beta \cdot \sum_{r_i \in U_t} \sum_{j \in N} \sum_{k \in [K]} pay_i \cdot (x^t_{i,j,m_i,k} + y^t_{i,j,m_i,k})\\
&- \beta \cdot \sum_{M_m \in \mathcal{M}} Update_cost(M_m, t) \cdot z_{m,t}\\
&- \beta \cdot \sum_{r_i \in U_t} \sum_{l_i \in \mathcal{C}_i} \sum_{j \in N} \sum_{k \in [K]} cost_{i,j,l_i,m_i,k} \cdot g^t_{i,l_i} \cdot x^t_{i,j,m_i,k}\\
&- \beta \cdot \sum_{r_i \in U_t} \sum_{l_i \in \mathcal{C}_i} \sum_{j \in N} \sum_{k \in [K]} cost_{i,j,l_i,m_i,k} \cdot g^t_{i,l_i} \cdot y^t_{i,j,m_i,k}\\
&- \beta \cdot \sum_{j \in N} \cdot \sum_{M_m \in \mathcal{M}} \sum_{k \in [K]} Init_cost(m,k) \cdot w^t_{j,m,k}\\
&- \beta \cdot \sum_{M_m \in \mathcal{M}} no_use_cost(M_m, t), \qquad (7)
\end{aligned}$$

where the optimization objective (7) is a weighted scalarization of user satisfaction and monetary net payoff (user payments minus the costs of resource consumptions) on services, where $\alpha$ and $\beta$ are coefficients to balance between the cumulative user satisfaction and the total user payoffs minus the total cost of various resources consumed on request admissions over the time horizon. Recall that $U_t$ is the set of incoming requests at time slot $t$. Let $X_t = \{x^t_{i,j,m,k} \mid r_i \in U_t, j \in N, M_m \in \mathcal{M}, k \in [K]\}$ and $Y_t = \{y^t_{i,j,m,k} \mid r_i \in U_t, j \in N, M_m \in \mathcal{M}, k \in [K]\}$. Within each time slot $t$, the first and second terms in the RHS of Eq. (7) are the cumulative user satisfaction and payoffs of admitted requests, the third term is the total updating cost of all updated models, the fourth term is the total cost of various resources consumed on admitted requests, the fifth term is the total cost of newly instantiated model instances of service models, and the last term $no_use_cost(M_m, t)$ is the total cost of all pre-deployed yet not used model instances that is defined as follows.

$$\begin{aligned}
&no_use_cost(M_m, t)\\
&= \sum_{j \in N} \sum_{k \in [K]} cost(M^{(k)}_m) \cdot (n^t_{j,m,k} - \left\lceil \frac{\sum_{r_i \in U_t}^{m_i = m} y^t_{i,j,m,k}}{L_{m,k}} \right\rceil), \qquad (8)
\end{aligned}$$

where $cost(M^{(k)}_m)$ is the resource consumption cost of an idle instance $M^{(k)}_m$ per time slot.

**Definition 1.** *Given an edge-cloud network $G = (N \cup \{c_0\}, E)$ with computing capacity $C_j$ for each cloudlet $j \in N$, a set $\mathcal{M}$ of primary service models in a remote cloud $c_0$ that are retrained continuously to maintain their freshness, each service model can be cached to a cloudlet in the MEC as a model instance for service provisioning through downloading from the cloud. Assume that each service model can have up to $K$ different model configuration instances with different amounts of resource allocated to each of such configurations. Given a time horizon $\mathbb{T}$, a set $U_t$ of requests arrived in the beginning of each time slot $t \in \mathbb{T}$, and each arrived request $r_i \in U_t$, represented by a tuple $\langle \mathcal{C}_i, vol_i, m_i, D_i \rangle$, must be responded by the admission or the rejection at time slot t, the QoE-aware service satisfaction maximization problem in $G$ is to maximize the long-term user satisfaction and payment on services while minimizing the total cost of various resources consumed on request admissions over time horizon $\mathbb{T}$, i.e., the objective is to maximize $\sum_{t=1}^{|\mathbb{T}|} OBJ(U_t, N, \mathcal{M}, X_t, Y_t, t)$, subject to computing capacity on each cloudlet and bandwidth capacity on each AP.*

## IV. NLIP FOR THE OFFLINE VERSION OF THE PROBLEM

In this section, we formulate a Nonlinear Integer Programming (NLIP) solution to the offline version of the QoE-aware service satisfaction maximization problem. Recall that each model instance $M^{(k)}_m$ consumes the amount $comp^{(k)}_m$ of computing resource.

Given the number $n^t_{j,m,k}$ of pre-deployed model instances $M^{(k)}_m$ in cloudlet $j$ at time slot $t \in \mathbb{T}$, the residual computing capacity on cloudlet $j$ at time slot $t$ is $C^t_j = C_j - \sum_{m=1}^{|\mathcal{M}|} \sum_{k=1}^{K} n^t_{j,m,k} \cdot comp^{(k)}_m$. The NLIP formulation is to

$$\text{Maximize} \sum_{t=1}^{|\mathbb{T}|} OBJ(U_t, N, \mathcal{M}, X_t, Y_t, t) \qquad (9)$$

s.t. (1)−(8),

$$AoI(M_m, t) = 1 + (1 - z_{m,t}) \cdot AoI(M_m, t-1), \quad \forall M_m \in \mathcal{M}, \forall t \in \mathbb{T}, \qquad (10)$$

$$\sum_{m \in \mathcal{M}} \sum_{k \in [K]} w^t_{j,m,k} \cdot comp^{(k)}_m \le C^t_j, \forall j \in N, \forall t \in \mathbb{T}, \qquad (11)$$

$$\sum_{r_i \in U_t} x^t_{i,j,m,k} \le L_{m,k} \cdot w^t_{j,m,k}, \forall j \in N, \forall M_m \in \mathcal{M}, \forall k \in [K], \forall t \in T, \qquad (12)$$

$$\sum_{l_i \in \mathcal{C}_i} \sum_{j=1}^{|N|} \sum_{k=1}^{K} (d_{i,j,l_i,m_i,k} + init_{m_i,k}) \cdot g^t_{i,l_i} \cdot x^t_{i,j,m_i,k} \le D_i, \quad \forall r_i \in U_t, \forall t \in \mathbb{T}, \qquad (13)$$

$$\sum_{l_i \in \mathcal{C}_i} \sum_{j=1}^{|N|} \sum_{k=1}^{K} d_{i,j,l_i,m_i,k} \cdot g^t_{i,l_i} \cdot y^t_{i,j,m_i,k} \le D_i, \ \forall r_i \in U_t, \forall t \in \mathbb{T}, \qquad (14)$$

$$\sum_{r_i \in U_t} y^t_{i,j,m,k} \le L_{m,k} \cdot n^t_{j,m,k}, \quad \forall j \in N, \forall m \in \mathcal{M}, \forall k \in [K], \forall t \in \mathbb{T}, \qquad (15)$$

$$\sum_{j \in N} \sum_{M_m \in \mathcal{M}} \sum_{k \in [K]} (x^t_{i,j,m,k} + y^t_{i,j,m,k}) \le 1, \quad \forall r_i \in U_t, \forall t \in \mathbb{T}, \qquad (16)$$

$$\sum_{r_i \in U_t} g^t_{i,l_i} \cdot \sum_{j \in N} \sum_{m \in \mathcal{M}} \sum_{k \in [K]} (x^t_{i,j,m,k} + y^t_{i,j,m,k}) \le L_{l_i}, \quad \forall l_i \in N, \forall t \in \mathbb{T}, \qquad (17)$$

$$g^t_{i,l_i} = 0, \text{ if } l_i \notin \mathcal{C}_i, \forall r_i \in U_t, \forall t \in \mathbb{T} \qquad (18)$$

$$x^t_{i,j,m,k} = 0, \ y^t_{i,j,m,k} = 0, \text{ if } m \ne m_i, \forall r_i \in U_t, \forall j \in N, k \in [K], \forall t \in \mathbb{T}, \qquad (19)$$

$$x^t_{i,j,m,k} \in \{0,1\}, \forall r_i \in U_t, \quad \forall j \in N, m \in \mathcal{M}, k \in [K], \forall t \in \mathbb{T}, \qquad (20)$$

$$y^t_{i,j,m,k} \in \{0,1\}, \forall r_i \in U_t, \quad \forall j \in N, m \in \mathcal{M}, k \in [K], \forall t \in \mathbb{T}, \qquad (21)$$

Authorized licensed use limited to: CITY UNIV OF HONG KONG. Downloaded on November 01,2026 at 03:34:31 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2026.3747852



$$z_{m,t} \in \{0,1\},\ \forall M_m \in \mathcal{M}, \forall t \in \mathbb{T}, \tag{22}$$

$$g^t_{i,l_i} \in \{0,1\},\ \forall r_i \in U_t, \forall l_i \in N, \forall t \in \mathbb{T}, \tag{23}$$

$$w^t_{j,m,k} \in \mathbb{Z}_{\geq 0}, \forall j \in N, \forall M_m \in \mathcal{M}, \forall k \in [K], \forall t \in T. \tag{24}$$

where Constraint (10) ensures that the AoIs of all model instances of $M_m$ in the MEC network are updated, i.e., if they are updated, their AoIs become 1, otherwise, their AoIs increase by 1. Constraint (11) ensures that the cumulative computing resource consumed by newly instantiated model instances in cloudlet $j$ does not exceed its residual capacity $C^t_j$. Constraint (12) ensures that the number of requests assigned to each newly instantiated instance does not exceed its batch processing capacity. Constraints (13) and (14) ensure that the end-to-end delay of an admitted request is no greater than its delay requirement. Constraint (15) ensures that the number of requests assigned to existing pre-deployed instances does not exceed their batch processing capacity. Constraint (16) ensures that each request $r_i$ is served by either an existing model instance or a newly instantiated model instance only. Constraint (17) ensures that the number of users uploading their request data via AP $l_i$ does not exceed the number $L_{l_i}$ of subchannels of AP $l_i$ at time slot $t$. Constraint (18) ensures that a request's device only can upload its request to an AP in its transmission range. It can be seen that variable $q(r_i,t)$ in the optimization objective (9) is nonlinear with variable $AoI(M_m,t)$, due to that the value of variable $AoI(M_m,t)$ is determined by the value of variable $z_{m,t}$. Also, the product of variables $q(r_i,t)$ and $(x^t_{i,j,m_i,k} + y^t_{i,j,m_i,k})$ is part of the optimization objective, the optimization objective thus is nonlinear. Furthermore, Constraints (13), (14), and (17) are nonlinear, too.

## V. Online algorithm for the QoE-aware service satisfaction maximization problem

In this section, we propose an online algorithm for the QoE-aware service satisfaction maximization problem.

The online algorithm first determines the number $n^t_{j,m,k}$ of pre-deployed model instances for each service model $M_m$ in each cloudlet $j \in N$, and determines whether the service model $M_m \in \mathcal{M}$ needs to update at time slot $t$, i.e., whether $z_{m,t}$ is set to 1. These decisions are learned from historical request traces for different service models. Specifically, for all requests of service model $M_m \in \mathcal{M}$, a digital twin $DT_m$ for model $M_m$ is deployed in the MEC, which records the details of each past request for the model including the issuing time slot, the location, the delay requirement, the admission status (accepted or rejected) of the request, and historical update time slots of the model $M_m \in \mathcal{M}$ in cloudlets. Then, the online algorithm assigns requests to cloudlets that host the selected number $n^t_{j,m,k}$ of model instances and $z_{m,t}$ for requested models. This is performed by a scheduler in the MEC network. The dynamic nature of resource consumption, coupled with random arrivals of requests pose significant challenges for the scheduler to make an optimal decision at each time slot. To this end, we cast the sequential model-management decision process as a Markov Decision Process (MDP) first. We then develop a Deep Reinforcement Learning (DRL) algorithm to find an approximate solution to the MDP that consists of (i) an agent that interacts with the environment and learns how to take actions by determining the values of $n^t_{j,m,k}$ and $z_{m,t}$ at each time slot $t$, and (ii) to admit requests in $U_t$ at time slot $t$ given the selected $n^t_{j,m,k}$ and $z_{m,t}$, we devise efficient algorithms for allocating limited sub-channels at each AP to requests under the coverage of the AP, followed by assigning channel-allocated requests to their requested model instances in cloudlets. We refer to this request admission procedure at time slot $t$ as the system scheduler that will be detailed later (in Section V-C).

### A. MDP formalization

The MDP consists of $(\mathcal{S}, \mathcal{A}, \mathcal{R})$, where $\mathcal{S}$ is the state set, $\mathcal{A}$ is the action set, and $\mathcal{R}$ is the reward function.

**State space.** At each time slot $t \in \mathbb{T}$, a $|\mathcal{M}|$-dimensional vector $\Phi(t)$ is used to keep the time slot of the previous updating of all service models, i.e., $\Phi(t) = (\Phi^{(t)}_1, \Phi^{(t)}_2, \ldots, \Phi^{(t)}_{|\mathcal{M}|})$, where $\Phi^{(t)}_m$ is the previous update time slot of model $M_m \in \mathcal{M}$ with $1 \leq \Phi^{(t)}_m \leq t-1$. Meanwhile, the computing resource consumption of cloudlet $j \in N$ at time slot $t-1$ is represented by a $|N|$-dimensional vector $\mathbf{l}(t-1)$ and $\mathbf{l}(t-1) = (l^{(t-1)}_1, l^{(t-1)}_2, \ldots, l^{(t-1)}_{|N|})$, where $l^{(t-1)}_j$ represents the utilization ratio of computing resource of cloudlet $j$ at time slot $t-1$. The requests in $U_t$ for each service model $M_m$ at time slot $t$ is represented by a $|\mathcal{M}|$-dimensional vector $\mathbf{I}(t) = (I^{(t)}_1, I^{(t)}_2, \ldots, I^{(t)}_{|\mathcal{M}|})$, where $I^{(t)}_m = |U^{(t)}_m|$, $U^{(t)}_m = \{r_i \mid r_i \in U_t, m_i = m\}$, and $|U^{(t)}_m|$ is the number of requests for model $M_m \in \mathcal{M}$ at time slot $t$. In addition, since the number of requests arriving at different time slots can be time-varying, we use a $\lambda(t) = |U_t|$ to represent the current request arrival intensity at time slot $t$. This term captures the overall workload level of the MEC system at the current time slot. Although $\lambda(t)$ can be derived from $\mathbf{I}(t)$ by $\lambda(t) = \sum_{m=1}^{|\mathcal{M}|} I^{(t)}_m$, we keep it as an explicit state component so that the MDP state directly reflects the current system workload. The historical information of requests for different service models prior to time slot $t$ is represented by a $|\mathcal{M}|$-dimensional vector $\mathbf{h}(t) = (h^{(t)}_1, h^{(t)}_2, \ldots, h^{(t)}_{|\mathcal{M}|})$, where $h^{(t)}_m = (p^{(t)}_m, pa^{(t)}_m)$ for service model $M_m \in \mathcal{M}$, $p^{(t)}_m$ is the ratio of the number of requests for service model $M_m$ to the number of requests for all models by time slot $t-1$ ( i.e., $p^{(t)}_m = \frac{\sum_{t'=1}^{t-1} |U^{(t')}_m|}{\sum_{t'=1}^{t-1} |U_{t'}|}$), and $pa^{(t)}_m$ is the ratio of the number of admitted requests for service model $M_m$ to the number of admitted requests for all models by time slot $t$ (i.e., $pa^{(t)}_m = \frac{\sum_{t'=1}^{t-1} |UA^{(t')}_m|}{\sum_{t'=1}^{t-1} |UA_{t'}|}$, $UA_{t'} = \{r_i \mid r_i \in U_{t'}, r_i \text{ is accepted}\}$ and $UA^{(t')}_m = \{r_i \mid r_i \in UA_{t'}, m_i = m\}$). Denote by $s_t \in \mathcal{S}$ the system state at time slot $t$, which is defined as follows.

$$s_t = (\Phi(t), \mathbf{l}(t-1), \mathbf{I}(t), \lambda(t), \mathbf{h}(t)), \tag{25}$$

where the dimension of $s_t$ is linear in the numbers of service models and cloudlets. In particular, $\Phi(t)$, $\mathbf{I}(t)$, $\mathbf{h}(t))$ have dimensions proportional to $|\mathcal{M}|$, $\lambda(t)$ is a scalar, and $\mathbf{l}(t-1)$ has dimension proportional to $|N|$.

**Action space.** With state $s_t$ at time slot $t \in \mathbb{T}$, the agent makes decisions by an optimal policy $\pi$ that includes (i) how

Authorized licensed use limited to: CITY UNIV OF HONG KONG. Downloaded on November 01,2026 at 03:34:31 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2026.3747852



many model instances of each service model need to pre-deploy; (ii) where they are deployed; and (iii) which existing model instances in cloudlets need to update. Specifically, the decision of pre-deployed model instances in each cloudlet $j \in N$ is represented by $n_j^{(t)} = (\vec{n}_{j,1}^{(t)}, \vec{n}_{j,2}^{(t)}, \ldots, \vec{n}_{j,|\mathcal{M}|}^{(t)})$, where the number of model instances of model $M_m \in \mathcal{M}$ deployed in cloudlet $j$ is represented by a $K$-dimensional vector $\vec{n}_{j,m}^{(t)} = (n_{j,m,1}^t, n_{j,m,2}^t, \ldots, n_{j,m,K}^t)$ with a non-negative integer $n_{j,m,k}^t$, and the pre-deployment of model instances is represented by a vector $\mathbf{n}(t)$ that is defined as follows.

$$\mathbf{n}(t) = (n_1^{(t)}, n_2^{(t)}, \ldots, n_{|N|}^{(t)}) \tag{26}$$

The model update decision at time slot $t \in \mathbb{T}$ is represented by $\mathbf{z}(t) = (z_{1,t}, z_{2,t}, \ldots, z_{|\mathcal{M}|,t})$, where binary variable $z_{m,t}$ indicates whether service model $M_m$ is updated in the beginning of time slot $t$ (i.e., $z_{m,t} = 1$ means that model $M_m$ is updated at time slot $t$). The action $\mathbf{a}^{(t)} \in \mathcal{A}$ of the agent at time slot $t$ is defined as $\mathbf{a}^{(t)} = (\mathbf{n}(t), \mathbf{z}(t))$, and its detailed explanation and implication will be given in the next subsection.

**State transition.** Once the agent takes action $\mathbf{a}^{(t)}$ at state $s_t$, the system state transits to the next state $s_{t+1}$. Specifically, the system scheduler is activated by solving the NLIP problem in (9) after obtaining action $\mathbf{a}^{(t)}$, and the NLIP problem now can be transformed into an Integer Linear Programming (ILP), as both $AoI(m,t)$ and $n_{j,m,k}^t$ at time slot $t$ are given by $\mathbf{a}^{(t)}$, and a feasible solution $\mathbb{S}_t$ to the ILP is obtained. Then, vectors $\mathbf{l}(t)$ and $\mathbf{h}(t)$ are updated based on $\mathbb{S}_t$, vectors $\Phi(t)$ is updated accordingly, and vector $\mathbf{I}(t)$ depends on the arrived requests at the next time slot.

**Reward function.** Denote by $R_t$ the reward obtained by action $\mathbf{a}^{(t)}$ at state $s_t$, which is

$$R_t = OBJ(U_t, N, \mathcal{M}, X_t, Y_t, t). \tag{27}$$

### *B. DRL algorithm*

In the following, we develop a Deep Q-Network (DQN)-based algorithm to proactively pre-deploy instances of different models and to choose service models for updating at each time slot [4], [24]. Compared with the Q-learning network, DQN utilizes a Deep Neural Network (DNN) to approximate the action-value function. The DRL algorithm proceeds as follows.

First, the DQN agent observes the current state $s_t$ and selects an action $\mathbf{a}^{(t)}$ using the $\epsilon$-greedy strategy. By taking action $\mathbf{a}^{(t)}$, the number $n_{j,m,k}^t$ of model instances $M_m^{(k)}$ for each model $M_m$ is pre-deployed in each cloudlet $j \in N$. Meanwhile, if $z_{m,t} = 1$, all model instances of model $M_m$ in the MEC are updated by downloading its primary model from the remote cloud and multicasting the downloaded model to all its instances in cloudlets.

Second, given the values of $n_{j,m,k}^t$ and $z_{m,t}$, the system scheduler determines the reward and state transition. Specifically, requests in $U_t$ are assigned to sub-channels of APs using `Algorithm 2`, and these sub-channel-allocated requests in $U_t'$ are then assigned to either newly instantiated or existing model instances in cloudlets, using `Algorithm 3`.

Finally, the agent receives a reward $R_t$ and the next state $s_{t+1}$. The experience $\mathbf{e}_t = (s_t, \mathbf{a}^{(t)}, R_t, s_{t+1})$ is added to a replay memory $\mathcal{B}$ with size $|\mathcal{B}|$ to train the DNN that approximates the optimal action-value function. By storing past transitions and randomly sampling mini-batches from them, the experience replay can further reduce temporal correlations among training samples. The reinforcement learning is to learn a policy to maximize the expected long-term cumulative reward. To this end, we define the optimal action-value function $Q^*(s,a)$ as follows.

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[\sum_{t'=t}^{\infty} \gamma^{t'-t} R_{t'} \mid s_t = s, \mathbf{a}^{(t)} = a\right], \tag{28}$$

which represents the maximum expected discounted return obtained by a policy $\pi = P(a|s)$ after making an observation $(s)$ and taking an action $(a)$.

Specifically, DQN employs two deep neural networks with identical architectures but different parameters: the *MainNet* (with the latest parameters $\theta$) for estimating current Q-values to select actions, and the *TargetNet* with parameters $\theta'$ (periodically copied from $\theta$) for computing stable target Q-values during the training process. Furthermore, the parameters $\theta$ of *MainNet* are learned from the randomly drawn samples of experiences $\mathcal{B}$, while the parameters of *TargetNet* are updated at every $C > 0$ steps by replicating the parameters of *MainNet*, where $C > 0$ is a predefined integer.

To improve training stability and learning performance, we combine Double Q-learning with a branching dueling network architecture [7], [8], [9], which is detailed as follows.

**Action representation and feasible action selection.** Both *MainNet* and *TargetNet* employ a branching dueling architecture that outputs a Q-value vector for each pre-deployment and model-update decision. Let $n_{\max}$ denote the maximum number of instances that each pre-deployment branch can select. For each cloudlet $j \in N$, model $M_m \in \mathcal{M}$, and configuration $k \in [K]$, the corresponding branch has $n_{\max}+1$ action values $Q_{j,m,k}^{\text{pre}}(s, v; \theta)$ with $v \in \{0, 1, \ldots, n_{\max}\}$. Each model-update branch has two action values $Q_m^{\text{upd}}(s, b; \theta)$ with $b \in \{0, 1\}$. Consequently, *MainNet* contains $|N||\mathcal{M}|K + |\mathcal{M}|$ branches, thereby avoiding enumerating all combinations of pre-deployments and update decisions. During exploitation, *MainNet* outputs $n_{\max} + 1$ Q-values for each pre-deployment branch, corresponding to the candidate instance counts from 0 to $n_{\max}$. If the candidate with the maximum Q-value is independently selected from each branch, the resource budget on a cloudlet may be violated. Therefore, constrained action selection on each cloudlet is performed through dynamic programming (DP), and one candidate instance count from each pre-deployment branch for each cloudlet is selected that maximizes the sum of the corresponding branch Q-values without violating the resource constraint on the pre-deployment. Such decisions for all cloudlets then form the pre-deployment action $\mathbf{n}(t)$. Meanwhile, each model updating decision then is determined by choosing the action with the maximum Q-value within its branch. During the $\epsilon$-greedy exploration, a pre-deployment action is constructed incrementally by randomly adding instances with sufficient resource

Authorized licensed use limited to: CITY UNIV OF HONG KONG. Downloaded on November 01,2026 at 03:34:31 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2026.3747852



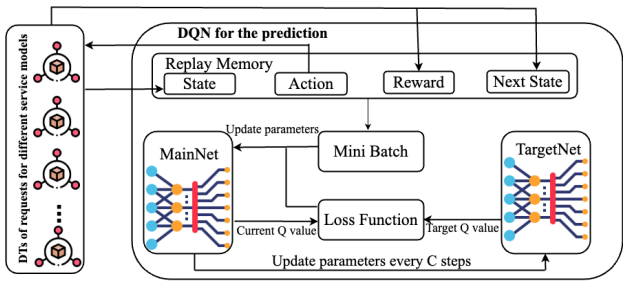



Fig. 2. A brief description of the DQN algorithm.

budgets while the update decisions are randomly sampled.

To compute the Double DQN target, let $\widehat{\mathbf{a}}^{(t+1)} = (\widehat{\mathbf{n}}(t+1), \widehat{\mathbf{z}}(t+1))$ be a feasible action chosen by *MainNet* at state $s_{t+1}$, where the value $\widehat{\mathbf{n}}(t+1)$ is obtained by dynamic programming with $\widehat{z}_{m,t+1} = \arg\max_{b\in\{0,1\}} Q_m^{\text{upd}}(s_{t+1}, b; \theta)$. The pre-deployment loss averages the temporal-difference errors over all $|N||\mathcal{M}|K$ pre-deployment branches, which is defined as

$$\mathcal{L}_{\text{pre}}(\theta) = \mathbb{E}_{\mathcal{B}}[\frac{1}{|N||\mathcal{M}|K} \sum_{j\in N} \sum_{m=1}^{|\mathcal{M}|} \sum_{k=1}^{K} \ell(R_t + \gamma Q_{j,m,k}^{\text{pre}}(s_{t+1}, \widehat{n}_{j,m,k}^{t+1}; \theta') - Q_{j,m,k}^{\text{pre}}(s_t, n_{j,m,k}^t; \theta))], \tag{29}$$

where $\gamma \in (0,1)$ is the reward discount factor, $\theta$ and $\theta'$ are the parameters of *MainNet* and *TargetNet* respectively, $\mathbb{E}_{\mathcal{B}}$ denotes the expectation over transitions sampled from the replay memory $\mathcal{B}$, and function $\ell(\cdot)$ is the Huber loss function to improve training stability.

The model updating loss is defined as

$$\mathcal{L}_{\text{upd}}(\theta) = \mathbb{E}_{\mathcal{B}}[\frac{1}{|\mathcal{M}|} \sum_{m=1}^{|\mathcal{M}|} \ell(R_t + \gamma Q_m^{\text{upd}}(s_{t+1}, \widehat{z}_{m,t+1}; \theta') - Q_m^{\text{upd}}(s_t, z_{m,t}; \theta))]. \tag{30}$$

The total loss of updating the parameters of *MainNet* is

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{pre}}(\theta) + \mathcal{L}_{\text{upd}}(\theta). \tag{31}$$

The detailed DQN algorithm is given in `Algorithm 1`. Specifically, the algorithm starts by initializing the parameters of replay memory $\mathcal{B}$, *MainNet*, and *TargetNet*, respectively. Then, in the beginning of each episode, a historical service-request trace is selected, the environment is initialized, and the initial state $s_1$ is derived. At each time slot $t \in \mathbb{T}$, one action $\mathbf{a}^{(t)}$ is selected by applying the branch-wise $\epsilon$-greedy strategy. Consequently, the reward $R_t$ defined in (27) and the next state $s_{t+1}$ are then obtained (Lines 8-9) after executing action $\mathbf{a}^{(t)}$. The experience $\mathbf{e}_t = (s_t, \mathbf{a}^{(t)}, R_t, s_{t+1})$, together with its terminal indicator, is then stored to the replay memory $\mathcal{B}$. The branch-wise DQN loss in Eq. (31) is used to update the parameters $\theta$ of *MainNet* and the parameters $\theta'$ of *TargetNet*, i.e., $\theta' \leftarrow \theta$ at every $C$ environment steps. The DQN algorithm is presented in Figure 2.

### C. Algorithms for request admissions at each time slot

Given the values of $n_{j,m,k}^t$ and $z_{m,t}$, the NLIP formulation of the QoE-aware service satisfaction maximization problem

**Algorithm 1** The DQN algorithm for proactive pre-deployment and update control.

**Input.** An edge-cloud network $G = (N \cup \{c_0\}, E)$, historical service request traces stored at DTs of requests for different service models in $\mathcal{M}$, a given discount factor $\gamma$ with $0 < \gamma < 1$, target-network update period $C > 1$, a given number of training episodes $\mathcal{E}$, time horizon $T$, mini-batch size $d$, an exploration rate $\epsilon$.

**Output.** Model parameters of *MainNet* $\theta$.

1: Initialize replay memory $\mathcal{B}$;
2: Initialize *MainNet* with random model parameters $\theta$;
3: Initialize *TargetNet* with model parameters $\theta'$; $\theta' \leftarrow \theta$;
4: **for** episode $ep \leftarrow 1$ to $\mathcal{E}$ **do**
5: Select a set of historical service request traces;
6: Initialize the environment and observe state $s_1$;
7: **for** $t \leftarrow 1$ to $T$ **do**
8: Select a feasible action $\mathbf{a}^{(t)}$ using the branch-wise $\epsilon$-greedy method;
9: The reward $R_t$ and the next state $s_{t+1}$ are obtained by taking action $\mathbf{a}^{(t)}$ by invoking `Algorithm 2` and `Algorithm 3`;
10: Store transition $\mathbf{e}_t \leftarrow (s_t, \mathbf{a}^{(t)}, R_t, s_{t+1})$ and its terminal indicator, to the replay memory $\mathcal{B}$;
11: **if** replay memory $\mathcal{B}$ contains at least $d$ transitions **then**
12: Sample a mini-batch of $d$ transitions uniformly from replay memory $\mathcal{B}$;
13: Compute the branch-wise loss $\mathcal{L}(\theta)$ by Eq. (31);
14: Update $\theta$ by minimizing $\mathcal{L}(\theta)$ through gradient descent;
15: **end if**
16: $\theta' \leftarrow \theta$, update *TargetNet* at every $C$ environment steps;
17: $s_t \leftarrow s_{t+1}$;
18: **end for**
19: **end for**
20: **return** The model parameters of *MainNet* $\theta$.

at time slot $t$ is reformulated as an ILP problem equivalently. However, the ILP is still intractable, instead it is decomposed into two sub-optimization problems: choosing requests to upload their inputs via APs by allocating sub-channels of the APs, and (ii) assigning each channel-allocated request to its requested model instance hosted by a cloudlet. Specifically, assume that $U_t$ is the set of arrived requests in the beginning of time slot $t$. We first develop an algorithm `Algorithm 2` for assigning requests in $U_t$ with sub-channels of APs, by finding a maximum matching in an auxiliary bipartite graph. Then, we assign channel-allocated requests to their requested model instances hosted by cloudlets, which is reduced to a maximum-profit Generalized Assignment Problem (GAP) [6], and solve the latter by applying `Algorithm 3`, which in turn delivers an approximate solution for the channel-allocated request assignment to cloudlets.

**Assigning requests to subchannels of APs.** We assign requests in $U_t$ to subchannels of APs as follows. We construct an auxiliary bipartite graph $G_{XY} = (X, Y, E_{XY})$, where $X$ is set of channel nodes with $X = \{x_{j,l} \mid j \in N, 1 \le l \le L_j\}$, $Y$ is the set of service requests with $Y = \{y_i \mid r_i \in U_t\}$, and $E_{XY}$ is the set of edges between nodes in $X$ and $Y$. There is an edge $(x_{j,l}, y_i) \in E_{XY}$ if request $r_i \in U_t$ is under the coverage of AP $j \in N$. Let $M$ be a maximum matching in $G_{XY}$. Then, for each matching edge $(x_{j,l}, y_i) \in M$, request $r_i$ will upload its input through subchannel $l$ of AP $j$ with $1 \le l \le L_j$ and $j \in N$. The proposed algorithm is detailed in `Algorithm 2`.

Authorized licensed use limited to: CITY UNIV OF HONG KONG. Downloaded on November 01,2026 at 03:34:31 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2026.3747852



**Algorithm 2** Algorithm for assigning requests in $U_t$ with sub-channels of APs at time slot $t$

**Input.** An MEC network $G = (N, E)$, a set $U_t$ of requests, and each AP $j \in N$ has $L_j$ sub-channels
**Output.** A subset $U'_t \subseteq U_t$ of requests, each allocated a sub-channel of an AP at time slot $t$
1: Construct a bipartite graph $G_{XY} = (X, Y, E_{XY})$, where $X$ is the set of sub-channels of APs, $Y$ is the set $U_t$ of requests, and there is an edge in $E_{XY}$ between request $r_i \in Y$ and each vertex/subchannel of AP $j$ in $X$ if $r_i$ is covered by AP $j$;
2: Find a maximum matching $M$ in graph $G_{XY}$;
3: **return** A subset $U_t$ of requests with each being allocated with a sub-channel of an AP at time slot $t$.

**Assigning sub-channel-allocated requests to cloudlets.** Let $U'_t = \{r_i \mid r_i \in U_t, \text{ and } (x_{j,l}, y_i) \in M\} \subseteq U_t$. We assign requests in $U'_t$ to their requested model instances hosted by different cloudlets within two stages. First, we determine the number $w^t_{j,m,k}$ of newly instantiated instances for each model configuration $M^{(k)}_m$ in each cloudlet $j$ with residual computing capacity. Second, given $n^t_{j,m,k}$ and $w^t_{j,m,k}$, we formulate the request assignment as a maximum-profit GAP with capacity constraints on cloudlets.

To determine $w^t_{j,m,k}$, we first identify the set $\mathcal{U}^t_{j,m,k}$ of requests that can be served by a newly instantiated instance of $M^{(k)}_m$ in cloudlet $j$, where $\mathcal{U}^t_{j,m,k} = \{r_i \in U'_t \mid m_i = m,\ d_{i,j,m,k} + init_{m,k} \leq D_i\}$. For each candidate pair $(j, m, k)$, we estimate its profit density as

$$\Delta_{j,m,k} = \frac{\sum_{r_i \in \mathcal{U}^t_{j,m,k}} (\alpha q(r_i, t) + \beta(pay_i - cost(i, j, m, k)))}{comp^{(k)}_m}. \tag{32}$$

For each cloudlet $j$, candidate model configurations are sorted in non-increasing order of $\Delta_{j,m,k}$. New instances are deployed greedily until the residual computing capacity $C^t_j$ is exhausted.

Given $w^t_{j,m,k}$, we construct two types of bins. One type corresponds to existing pre-deployed model instances. For each cloudlet $j \in N$, service model $M_m \in \mathcal{M}$, and configuration $k \in [K]$, we construct an existing-instance bin $B^{\mathrm{E}}_{j,m,k}$ with capacity $n^t_{j,m,k}$. Another type corresponds to candidate newly instantiated model instances. For each cloudlet $j \in N$, model $M_m \in \mathcal{M}$, and configuration $k \in [K]$, we construct a new-instance bin $B^{\mathrm{N}}_{j,m,k}$ with capacity $w^t_{j,m,k}$. The pre-deployed and newly instantiated instances occupy $comp^{(k)}_m n^t_{j,m,k}$ and $comp^{(k)}_m w^t_{j,m,k}$ computing resources, respectively.

For each request $r_i \in U'_t$ (an item to be packed to one bin) requesting for service model $M_{m_i}$, item $i$ can be assigned to one of the above bins by the following two cases.

Case 1. Item $i$ can be packed to a newly instantiated model instance (a bin) $B^{\mathrm{N}}_{j,m_i,k}$ if $d_{i,j,m_i,k} + init_{m_i,k} \leq D_i$ is met. The assignment consumes $1/L_{m_i,k}$ unit of the bin capacity, and the profit is $profit^{\mathrm{N}}(i, j, m_i, k) = \alpha \cdot q(r_i, t) + \beta \cdot (pay_i - cost(i, j, m_i, k))$. The instantiation cost of model instances will not be charged to requests, which instead is counted once when the number of newly instantiated model instances $w^t_{j,m,k}$ is instantiated at time slot $t$.

Case 2. Item $i$ is packed to an existing model instance (a bin) $B^{\mathrm{E}}_{j,m_i,k}$ if $d_{i,j,m_i,k} \leq D_i$ is satisfied. The profit is $profit^{\mathrm{E}}(i, j, m_i, k)$ with weight $\frac{1}{L_{m,k}}$, where $profit^{\mathrm{E}}(i, j, m_i, k) = \alpha \cdot q(r_i, t) + \beta \cdot (pay_i - cost(i, j, m_i, k))$. If item $i$ is assigned to a bin $B^{\mathrm{E}}_{j,m,k}$ with $m \neq m_i$, then this assignment results in a profit of zero with an infinite weight, implying this assignment is infeasible.

**Algorithm 3** Algorithm for assigning requests in $U'_t$ to model instances in cloudlets at time slot $t$

**Input.** An MEC network $G = (N, E)$, a set $U'_t$ of channel-allocated requests, and given values of $\mathbf{n}(t)$ and $\mathbf{z}(t)$
**Output.** A request assignment solution at time slot $t$
1: Initialize $S_t \leftarrow 0$, $\mathbb{S}_t \leftarrow \emptyset$, $x^t_{i,j,m,k} \leftarrow 0$, $y^t_{i,j,m,k} \leftarrow 0$, and $w^t_{j,m,k} \leftarrow 0$;
2: **for** each cloudlet $j \in N$ **do**
3: $C^{\mathrm{res}}_j \leftarrow C^t_j$;
4: **for** each service model $M_m \in \mathcal{M}$ and configuration $k \in [K]$ **do**
5: Find the candidate request set $\mathcal{U}^t_{j,m,k}$
6: Compute the profit density $\Delta_{j,m,k}$
7: **end for**
8: Sort all candidate pairs $(m, k)$ in non-increasing order of $\Delta_{j,m,k}$;
9: **for** each sorted pair $(m, k)$ **do**
10: $w^t_{j,m,k} \leftarrow \min\left\{\left\lceil \frac{|\mathcal{U}^t_{j,m,k}|}{L_{m,k}} \right\rceil, \left\lfloor \frac{C^{\mathrm{res}}_j}{comp^{(k)}_m} \right\rfloor\right\}$;
11: $C^{\mathrm{res}}_j \leftarrow C^{\mathrm{res}}_j - w^t_{j,m,k} comp^{(k)}_m$.
12: **end for**
13: **end for**
14: Construct a maximum-profit GAP instance;
15: **for** each request $r_i \in U'_t$ **do**
16: **for** each cloudlet $j \in N$ and configuration $k \in [K]$ **do**
17: **if** $d_{i,j,m_i,k} \leq D_i$ **then**
18: Add a feasible assignment option of $r_i$ to $B^{\mathrm{E}}_{j,m_i,k}$ with profit $profit^{\mathrm{E}}(i, j, m_i, k)$
19: **end if**
20: **if** $d_{i,j,m_i,k} + init_{m_i,k} \leq D_i$ and $w^t_{j,m_i,k} > 0$ **then**
21: Add a feasible assignment option of $r_i$ to $B^{\mathrm{N}}_{j,m_i,k}$ with profit $profit^{\mathrm{N}}(i, j, m_i, k)$
22: **end if**
23: **end for**
24: **end for**
25: Solve the resulting maximum-profit GAP approximately;
26: **for** each request $r_i \in U'_t$ **do**
27: **if** $r_i$ is assigned to bin $B^{\mathrm{E}}_{j,m_i,k}$ **then**
28: $y^t_{i,j,m_i,k} \leftarrow 1$;
29: **else if** $r_i$ is assigned to bin $B^{\mathrm{N}}_{j,m_i,k}$ **then**
30: $x^t_{i,j,m_i,k} \leftarrow 1$;
31: **end if**
32: **end for**
33: $\mathbb{S}_t \leftarrow \{(x^t_{i,j,m,k}, y^t_{i,j,m,k}, w^t_{j,m,k}) \mid r_i \in U'_t, j \in N, M_m \in \mathcal{M}, k \in [K]\}$;
34: Calculate the objective value $S_t$ by Eq. (7);
35: **return** $\mathbb{S}_t$ and $S_t$.

The proposed algorithm for assigning channel-allocated requests to cloudlets is detailed in `Algorithm` 3.

### *D. Online algorithm*

With the DQN module for pre-deployments of the number of instances of each service model in each cloudlet and model updating decisions by `Algorithm` 1, we now propose an online algorithm for the QoE-aware service satisfaction maximization problem as follows.

Let $\mathbb{S}$ be the solution to the problem and $S$ the value of $\mathbb{S}$, respectively. At each time slot $t \in \mathbb{T}$, the agent observes the current state $s_t$. At state $s_t$, the agent uses the trained

Authorized licensed use limited to: CITY UNIV OF HONG KONG. Downloaded on November 01,2026 at 03:34:31 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2026.3747852



**Algorithm 4** Online algorithm for the QoE-aware service satisfaction maximization problem.

**Input.** An edge-cloud network $G = (N \cup \{c_0\}, E)$, a set $U_t$ of arrived requests at each time slot $t$ over time horizon $\mathbb{T}$, and the trained *MainNet* by `Algorithm` 1.

**Output.** A solution $\mathbb{S}$ to the problem to maximize the objective function (9).

Initialize the environment and observe the initial state $s_1$;
$\mathbb{S} \leftarrow \emptyset$; $S \leftarrow 0$; /* the problem solution and the value of the solution */
**for** $t \leftarrow 1$ to $T$ **do**
  Feed state $s_t$ into the trained *MainNet* to obtain the pre-deployment and model-update branch Q-values;
  Find a feasible pre-deployment decision $\mathbf{n}(t)$ using the DP for each cloudlet;
  Select the candidate with the maximum Q-value from its model-update branch to obtain $z_{m,t}$ for each model $M_m \in \mathcal{M}$;
  $\mathbf{a}^{(t)} \leftarrow (\mathbf{n}(t), \mathbf{z}(t))$;
  Take action $\mathbf{a}^{(t)}$, and then assign requests in $U_t$ to AP sub-channels by invoking `Algorithm` 2;
  Let $U'_t$ be the set of sub-channel assigned requests. Assign requests in $U'_t$ to their requested model instances in cloudlets by invoking `Algorithm` 3;
  A solution $\mathbb{S}_t$ with reward $R_t$ is obtained at each time slot $t$;
  $\mathbb{S} \leftarrow \mathbb{S} \cup \mathbb{S}_t$; $S \leftarrow S + R_t$;
  Observe the next state $s_{t+1}$, and $s_t \leftarrow s_{t+1}$;
**end for**
**return** $\mathbb{S}$.

*MainNet* to generate a feasible action $\mathbf{a}^{(t)} = (\mathbf{n}(t), \mathbf{z}(t))$. With action $\mathbf{a}^{(t)}$, the sub-channel allocation for requests in $U_t$ is obtained by invoking `Algorithm` 2, followed by assigning these channel-allocated requests to their requested model instances in cloudlets by invoking `Algorithm` 3. Having requests admitted at time slot $t$, the agent interacts with the environment; the value of $\mathbb{S}$ is updated and the next state $s_{t+1}$ is observed (i.e., $s_t \leftarrow s_{t+1}$).

The feasibility of a solution does not depend on the accuracy of the branch Q-value estimates. The constrained action-selection procedure in Section V-B ensures that the pre-deployment decisions satisfy cloudlet computing-resource constraints, whereas `Algorithm` 2 and `Algorithm` 3 further enforce the AP sub-channel capacity, cloudlet computing-resource capacity, instance capacity, and end-to-end delay constraints for the admission of each request. In other words, inaccurate Q-value estimates may lead `Algorithm` 4 to produce a suboptimal solution, but the solution it delivers is feasible, and none of the constraints is violated. The detailed description of the algorithm is given in `Algorithm` 4.

### *E. Algorithm analysis*

**Lemma 1.** *Given an edge-cloud network $G = (N \cup \{c_0\}, E)$ with $L_j$ sub-channels of each AP $j \in N$, a set $U_t$ of incoming requests at time slot $t \in \mathbb{T}$, there is an algorithm,* `Algorithm` *2, for the sub-channel allocation problem, which takes $\mathcal{O}(\sqrt{|N|+|U_t|} \cdot |N| \cdot |U_t|)$ time per time slot $t$.*

*Proof.* We show that the solution delivered by `Algorithm` 2 is feasible. In the constructed bipartite graph $G_{XY}$, each request is matched with at most one sub-channel via an AP. For each AP $j$, there are at most $L_j$ vertices in $G_{XY}$, which indicates that there are at most $L_j$ matching edges associated with AP $j$. Thus, the number of requests to upload their input data through AP $j$ will not be greater than $L_j$. The solution delivered by the proposed algorithm is feasible. The construction of $G_{XY}$ takes $\mathcal{O}(|X| \cdot |Y|)$ time, and finding a maximum matching in $G_{XY}$ takes $\mathcal{O}(\sqrt{|X|+|Y|} \cdot |X| \cdot |Y|)$ time. Since the number of subchannels in an AP is a constant, the time complexity of `Algorithm` 2 is $\mathcal{O}(\sqrt{|X|+|Y|} \cdot |X| \cdot |Y|) = \mathcal{O}(\sqrt{|N|+|U_t|} \cdot |N| \cdot |U_t|)$. □

**Lemma 2.** *Given an edge-cloud network $G = (N \cup \{c_0\}, E)$, for each time slot $t \in \mathbb{T}$, a set $U'_t$ ($\subseteq U_t$) of incoming requests, a given $|\mathcal{M}|$-dimensional vector $\mathbf{z}(t)$ for all inference models $M_m \in \mathcal{M}$ and the number $n^t_{j,m,k}$ of pre-deployed model instances $M^{(k)}_m$ in each cloudlet $j \in N$ at time slot $t$,* `Algorithm` *3 delivers a feasible assignment of requests in $U'_t$ to their requested model instances in cloudlets. Given the numbers $w^t_{j,m,k}$ of newly instantiated instances determined by the greedy process in the first stage, the second-stage request assignment is reduced to a maximum-profit GAP and admits a $\frac{1}{2+\delta}$-approximation ratio, where $\delta$ is a constant with $0 < \delta < 1$. The time complexity of* `Algorithm` *3 is $\mathcal{O}(|U_t| \cdot K \cdot |N| \cdot |\mathcal{M}| \cdot \log \frac{1}{\delta} + \frac{K \cdot |N| \cdot |\mathcal{M}|}{\delta^4})$.*

*Proof.* `Algorithm` 3 first determines the numbers $w^t_{j,m,k}$ of newly instantiated instances by the greedy opening step. During this step, new instances are opened only under the residual computing capacity of each cloudlet. Thus, the newly instantiated instances satisfy the cloudlet resource constraints. Given $w^t_{j,m,k}$, the remaining request assignment is a maximum-profit GAP. By applying the approximation algorithm in [6], the second-stage assignment achieves a $\frac{1}{2+\delta}$ approximation ratio with respect to the optimal assignment.

The time complexity of `Algorithm` 3 is analyzed as follows. Computing the candidate sets and the greedy opening decisions takes $\mathcal{O}(|U_t| \cdot K \cdot |N| \cdot |\mathcal{M}|)$ time. Given $w^t_{j,m,k}$, the GAP contains at most $2|N| \cdot K \cdot |\mathcal{M}|$ bins and $|U'_t|$ items. The approximation algorithm for the maximum-profit GAP takes $\mathcal{O}(|U'_t| \cdot K \cdot |N| \cdot |\mathcal{M}| \cdot \log \frac{1}{\delta} + \frac{K \cdot |N| \cdot |\mathcal{M}|}{\delta^4})$. Since $|U'_t| \leq |U_t|$ and the GAP-solving step dominates the greedy opening step, the claimed time complexity follows. □

**Theorem 1.** *Given an edge-cloud network $G = (N \cup \{c_0\}, E)$, a given time horizon $\mathbb{T}$, a set $U_t$ of service requests at time slot $t$, and a set $\mathcal{M}$ of service models with $K$ different configuration instances for each model, there is an online algorithm,* `Algorithm` *4, for the QoE-aware service satisfaction maximization problem, which delivers a feasible solution, and takes $\mathcal{O}(T_{\text{inf}} + \sqrt{|N|+|U_t|} \cdot |N| \cdot |U_t| + |U_t| \cdot |N| \cdot K \cdot |\mathcal{M}| \cdot \log \frac{1}{\delta} + \frac{|N| \cdot K \cdot |\mathcal{M}|}{\delta^4})$ time per time slot $t$, assuming that $T_{\text{inf}}$ denotes the computational cost of one forward pass through MainNet, where $\delta$ is a constant with $0 < \delta < 1$.*

*Proof.* It can be seen that the solution delivered by `Algorithm` 4 is feasible, which is ensured by Lemma 2.

The rest is to analyze its time complexity. The initial DQN network training is determined by multiple parameters of the system such as the number $|\mathcal{M}|$ of service models, the network size $|N|$, the size of historical service request traces, and the

Authorized licensed use limited to: CITY UNIV OF HONG KONG. Downloaded on November 01,2026 at 03:34:31 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2026.3747852



number $K$ of configuration instances of each model. The DQN however is trained only once in an offline manner. At each time slot, generating the action requires one forward pass through *MainNet*, followed by the feasible action-selection procedure in Section V-B. Denote by $T_{\text{inf}}$ the cost of one forward pass. For a fully connected `MainNet` with layer width $H_0, H_1, \ldots, H_L$, $T_{\text{inf}} = \mathcal{O}(\sum_{\ell=1}^{L} H_{\ell-1}H_\ell)$. Given the fixed value $n_{\max}$ and the fixed number of resource-budget layers, it takes $\mathcal{O}(|N||\mathcal{M}|K)$ time to find the maximum value among the $|\mathcal{M}|K$ pre-deployment branches of each cloudlet, by applying the DP procedure, while it takes $\mathcal{O}(|\mathcal{M}|)$ time to make model-update decisions, the time complexity of the action-selection procedure thus is no greater than that of `Algorithm` 3. For admissions of requests in $U_t$ at time slot $t$, the time complexity of `Algorithm` 4 is determined by invoking `Algorithm` 2 and `Algorithm` 3, and their time complexities are given in Lemma 1 and 2, respectively. The theorem then follows. □

## VI. Performance Evaluation

In this section, we evaluate the performance of the proposed online algorithm for the QoE-aware service satisfaction maximization problem. We also investigate impacts of important parameters on the performance of the proposed algorithm.

### A. Experimental environment settings

We considered an edge-cloud network consisting of 20 APs that is generated by NetworkX [25]. The normalized computing capacity of each cloudlet is randomly drawn from [4,000, 8,000] resource units. The bandwidth on each AP is randomly chosen in [20, 40] MHz, and each AP is randomly assigned 5 to 10 sub-channels. The signal-to-noise ratio of each device is randomly chosen from 10dB to 30dB [39]. User devices can be located at different locations at different time slots, and a device can be covered from 2 to 8 APs at a location. The cost $\xi$ per unit bandwidth resource (Mbps) is \$0.0001 [29]. The transmission delay of one MB data along a link $e \in E$ is randomly drawn in [0.02, 0.05] milliseconds, and the transfer cost $\zeta_e$ per unit data along a link is randomly drawn from \$0.01 to \$0.02 per MB [36]. We assume that there are ten service models with each having six instance configurations, i.e., $|\mathcal{M}| = 10$ and $K = 6$. For each service model, the resource demands of its $K$ configuration instances are within $[200, 400]$ resource units [39] and arranged in increasing order of $k$, and their corresponding processing delays are within $[10, 100]$ ms and assigned in decreasing order of $k$ with $1 \leq k \leq K$. Consequently, for any two configurations with indices $k$ and $k'$, $comp_m^{(k)} < comp_m^{(k')}$ and $\tau_{m,k} > \tau_{m,k'}$ with $k < k'$. A model configuration with more resource units always has a lower processing delay. The inference cost $\varphi_{j,m,k}$ is in the range of \$[0.40, 2.80]. By default, each model instance can simultaneously serve up to eight requests, i.e., $L_{m,k} = 8$. The cost $Init_cost(m,k)$ of instantiating a new model instance is set at $Base_m + k * \kappa$, where $Base_m$ is a constant in the range of \$[2, 14] [22], which depends on the service type $M_m$, and $\kappa$ is a coefficient for a different model configuration instance $k$ with $1 \leq k \leq K$.

TABLE II
Statistics of the Azure-derived request arrival sequences.

| Statistic | Value |
|---|---|
| Target average arrivals | 200 requests per slot |
| Rescaled arrivals | 144 – 406 requests per slot |
| Standard deviation | 20.8 |
| Peak-to-average ratio | 2.03 |

Similarly, the cost of an idle model instance is randomly chosen within \$[0.20, 1.40]. The instantiation delay $init_{m,k}$ of a new model instance is randomly drawn in the range of [600, 800] milliseconds. The default value of the base $a$ for measuring the freshness of a service model is set at 2. The values of $\alpha$ and $\beta$ in the objective function are set at 0.8 and 0.2 by default, respectively. Each request $r_i$ requests a service model $M_{m_i}$ with the input volume randomly drawn from [5, 10] MB [20]. Its delay requirement $D_i$ is randomly drawn between 2,000 and 3,000 ms. The payment $pay_i$ is determined by its requested service model $M_{m_i}$ and the delay-requirement level, where the delay-requirement range is divided into four levels [37].

To capture time-varying request arrivals, we use the Azure public dataset [10] as the temporal source of request arrivals. Since the raw request counts in the Azure dataset are collected from a large-scale platform, its raw counts are not directly used in our MEC-scale simulations. Instead, we make use of the temporal variation pattern of request arrivals. Specifically, we normalize the arrival pattern from the Azure dataset and rescale it according to a target average request intensity $\overline{|U_t|}$, which denotes the target average number of requests per time slot after rescaling. In this way, $|U_t|$ varies across time slots while preserving the temporal fluctuation pattern of the Azure dataset, and its average over the time horizon is approximately $\overline{|U_t|}$. Unless otherwise specified, the average number of requests per time slot $\overline{|U_t|}$ is set at 200. Table II summarizes the temporal characteristics of the sequence of generated requests under the default setting. The number of time slots for a given time horizon $\mathbb{T}$ is 100. Table III summarizes the key hyperparameter settings used in `Algorithm` 1.

To evaluate the DRL-based approach in `Algorithm` 4, we compared the algorithm with the three baselines that adopt different model management and request admission strategies. For model management, we considered the following two mechanisms: sliding-window instance management (`SWIM`), and the LSTM-based predictive pre-deployment (`LPP`), respectively.

`SWIM` is a sliding window based method [37]: Given a window length $W \geq 1$, a counter is associated with each existing instance to count its non-usages in the past consecutive time slots. If no request is assigned to the instance at the current time slot, its counter increases by 1; otherwise its counter reduces to 0. When its counter reaches the window length $W$, the model instance is revoked at the end of the current time slot. Since each instance $M_m^{(k)}$ can serve up to $L_{m,k} \geq 1$ requests at each time slot, a model instance is regarded as used if at least one request is assigned to it at the current time slot. By default, we set $W = 2$.

`LPP` outputs $n_{j,m,k}^t$ by learning temporal demand patterns

Authorized licensed use limited to: CITY UNIV OF HONG KONG. Downloaded on November 01,2026 at 03:34:31 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2026.3747852



TABLE III
THE MAIN HYPERPARAMETERS OF THE DQN MODULE.

| Parameter | Value / setting |
|---|---|
| Network architecture | Fully connected DNN |
| Hidden layers | 3 |
| Hidden units per layer | $[512, 256, 128]$ |
| Activation function | ReLU |
| Optimizer | Adam |
| Learning rate | $10^{-5}$ |
| Maximum pre-deployment value $n_{\max}$ | 4 |
| Discount factor $\gamma$ | 0.995 |
| Replay memory size $\lvert\mathcal{B}\rvert$ | 50,000 |
| Mini-batch size | 128 |
| Target update period $C$ | 1000 steps |
| Initial exploration rate $\epsilon_0$ | 1.0 |
| Final exploration rate $\epsilon_{\min}$ | 0.05 |
| Exploration decay | Exponential decay |
| Training episodes | 600 |

from request observations in previous time slots. Specifically, the LSTM predictor predicts the future request demands for each model type, and the predicted demands are then converted into the required numbers of model instances according to their service capacities $L_{m,k}$. The predicted model instances are placed to cloudlets, subject to computing resource capacities on cloudlets. In the LSTM predictor implementation, its input sequence length and prediction window are set at 8 and 3, respectively. The predictor contains one LSTM layer with a 32-dimension hidden state, and the Adam optimizer with learning rate of 0.01 and the Huber loss function are adopted.

For request admissions, we adopted several baselines that utilize a greedy rule based on the contribution of admitting a request to the optimization objective. That is, if request $r_i \in U_t$ is admitted to its requested model instance $M_{m_i}^{(k)}$ in cloudlet $j$, then the contribution of request $r_i$ to the objective function is

$$\sum_{j\in N}\sum_{k\in[K]} (\alpha \cdot q(r_i,t) + \beta \cdot (pay_i - cost_{i,j,l_i,m_i,k})) \cdot (x^t_{i,j,m_i,k} + y^t_{i,j,m_i,k}). \quad (33)$$

In this paper, three baselines against the proposed algorithm are proposed, which are defined as follows.

`Heur_1` adopts `SWIM` to manage pre-deployed instances, assigns binary variable $z_{m,t}$ with 0/1 randomly, and examines requests in their arrival order. For each request, it randomly selects a cloudlet and admits the request if an existing model instance that it requested is available; otherwise a new model instance it requested is instantiated in a randomly selected cloudlet with adequate computing resource for the instance.

`Heur_2` makes use of both `SWIM` and random $z_{m,t}$, and assigns feasible request-instance pair with the largest request-level contribution in (33) greedily.

`Heur_3` adopts `LPP` to predict the values of both $n^t_{j,m,k}$ and $z_{m,t}$, then admits requests to feasible pre-deployed instances greedily by their contributions to the optimization objective.

Each reported value in figures is averaged over 25 network topologies generated by NetworkX [25] with the same network size. The running time of an algorithm is obtained on a desktop equipped with an Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz, with 62GB RAM. The aforementioned parameters will be adopted by default unless otherwise specified.

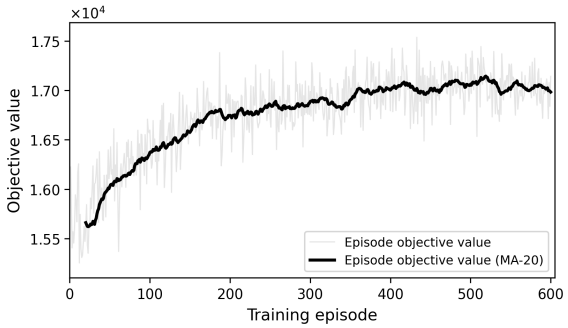



Fig. 3. The DQN training performance of `Algorithm` 1.

### B. Performance of different algorithms for the QoE-aware service satisfaction maximization problem

We first evaluated the performance of `Algorithm` 1 in the DQN training, Fig. 3 by plotting its performance curve under the default setting in Table III. It can be observed that the moving average of the objective value increases rapidly in the early training stage and gradually stabilizes after about 400 episodes. This shows that the DQN module can learn an effective model management policy through interactions with the MEC environment and `Algorithm` 1 exhibits stable training behavior.

Having the trained DQN, we evaluate the performance of the online algorithm, `Algorithm` 4, for the QoE-aware service satisfaction maximization problem against the three baselines by varying the target average number $\overline{|U_t|}$ of requests per time slot and network size $|N|$.

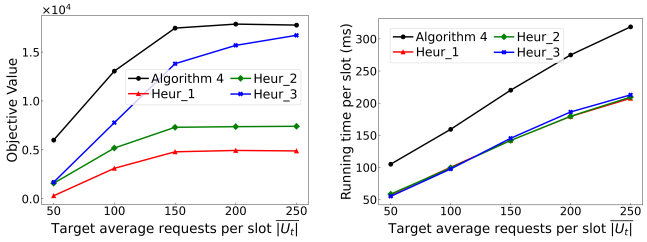
We varied the target average number $\overline{|U_t|}$ of requests per slot while fixing the network size. Since the network size remains unchanged, a larger $\overline{|U_t|}$ indicates a heavier network load per cloudlet on average. Fig. 4 shows the objective value and running time of different algorithms. As shown in Fig. 4(a), `Algorithm` 4 consistently achieves the highest objective value across all examined request intensities. Its objective value increases substantially as $\overline{|U_t|}$ increases from 50 to 150 and then remains relatively stable under heavier workloads. This is because more requests provide more opportunities for service admission and utility improvement when the system is not saturated, while the limited computing and channel resources gradually become the bottleneck as the request intensity further increases.

We studied the performance of different algorithms by varying the network size $|N|$ from 10 to 50. It can be observed from Fig. 5(a) that `Algorithm` 4 outperforms all other comparison algorithms under different network sizes. Its objective value increases with $|N|$ from 10 to 40. This is because the available computing resources and AP channels increase when the network size increases, enabling more requests to be admitted. However, when $|N|$ reaches 50, the objective values of all algorithms slightly decrease. This is mainly because the marginal benefit of adding more cloudlets becomes limited

Authorized licensed use limited to: CITY UNIV OF HONG KONG. Downloaded on November 01,2026 at 03:34:31 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

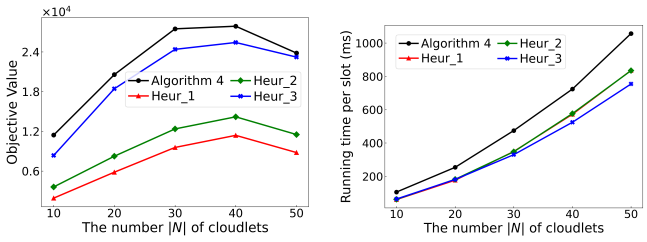
This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2026.3747852





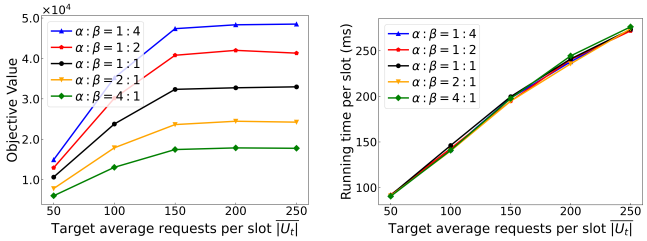

(a) The performance (b) The running time

Fig. 4. Performance of different algorithms by varying the target average number $|\overline{U_t}|$ of requests per time slot.




(a) The performance (b) The running time

Fig. 5. Performance of different algorithms by varying the number $|N|$ of cloudlets.




(a) The performance (b) The running time

Fig. 6. Impact of coefficients $\alpha$ and $\beta$ on the performance of `Algorithm 4` under different target average numbers $|\overline{U_t}|$ of requests per time slot.

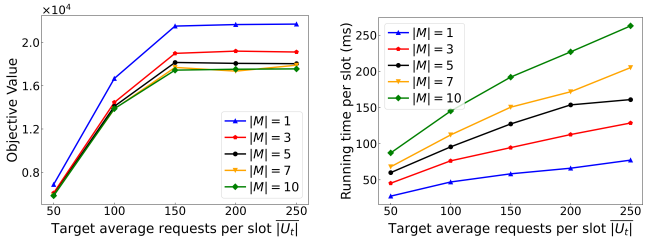
under a fixed request intensity. A larger network may introduce more pre-deployed instances and management cost, while the total request demand does not increase proportionally, resulting in less effective resource utilization.

### C. Impacts of parameters on algorithm performances

We studied the impacts of key parameters on the performance of the proposed algorithm for the problem as follows.

**Impact of coefficients $\alpha$ and $\beta$.** We investigated impacts of coefficients $\alpha$ and $\beta$ on the performance of `Algorithm 4`, by setting the ratio of $\frac{\alpha}{\beta}$ at $\frac{1}{4}$, $\frac{1}{2}$, 1, 2 and 4, respectively. A larger ratio places a greater emphasis on user satisfaction associated with model freshness, whereas a smaller ratio places a greater emphasis on monetary net payoff. It can be seen from Fig. 6(a) that an increase in the ratio of $\alpha$ to $\beta$ results in a lower objective value, indicating that the payoff term outweighs the user service satisfaction of the objective function. In addition, with the increase on the number of user requests per time slot, the objective value first grows, and then gradually stabilizes when the number of requests reaches 150. This is due to the limited computing resource in the MEC network constrains the number of request admissions. Fig. 6(b) shows that the impact of the coefficient ratio on the running time of the algorithm is minor.

**Impact of the number $|\mathcal{M}|$ of models.** We evaluated the impact of the number of models on the performance of `Algorithm 4`, by choosing $|\mathcal{M}| = 1, 3, 5, 7$, and 10. It can be seen from Fig. 7(a) that an increase in the number $|\mathcal{M}|$ of service models will generally lead to a lower objective value when the number of requests per time slot is fixed. This is because the limited computing resources are shared by more model types, which leaves fewer pre-deployed instances for each type and reduces the chance that a request can be served by a feasible existing instance. In addition, requests for less popular models may trigger new instance instantiation, which incurs extra overhead. Fig. 7(b) shows that the running time increases with both $|\mathcal{M}|$ and the request intensity because of the enlarged decision space for model management and request assignment.




(a) The performance (b) The running time

Fig. 7. Impact of the number $|\mathcal{M}|$ of service models on the performance of `Algorithm 4` under different target average numbers $|\overline{U_t}|$ of requests per time slot.

## VII. Conclusion

In this paper, we studied QoE-aware inference services in an edge-cloud computing network, by formulating a QoE-aware service satisfaction maximization problem over a given time horizon. We first formulated an NLIP solution to the offline version of the problem. To address the online setting with time-varying request arrivals, we developed a DRL-based framework that models the sequential decision-making process as an MDP. The DQN module learns how many model instances should be pre-deployed in cloudlets and which service models should be updated at each time slot. Given these decisions, the channel allocation is solved by maximum matching, and channel-allocated requests are further assigned to cloudlets through a GAP-based assignment algorithm. We finally evaluated the performance of the proposed algorithm by simulations. Simulation results show that the proposed algorithm is superior to the three comparison counterparts.

## References

[1] X. Ai, W. Liang, and C. Liu. Joint optimization of model retraining and inference services in DT-assisted edge computing. *IEEE Transactions on Networking*, vol. 34, pp. 1804–1819, 2026.

[2] N. Akhtar, A. Raza, V. Ishakian, and I. Mattar. COSE: Configuring serverless functions using statistical learning. *Proc. of INFOCOM'20*, pp. 129 – 138, IEEE, 2020.

[3] J. Chen and X. Ran. Deep learning with edge computing: a review. *Proc. of IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[4] X. Chen, S. Hu, C. Yu, Z. Chen, and G. Min. Real-time offloading for dependent and parallel tasks in cloud-edge environments using deep reinforcement learning. *IEEE Transactions on Parallel and Distributed Systems*, vol. 35, no. 3, pp. 391 – 404, 2024.

Authorized licensed use limited to: CITY UNIV OF HONG KONG. Downloaded on November 01,2026 at 03:34:31 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Transactions on Mobile Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TMC.2026.3747852




[5] Z. Chen, S. Zhang, Z. Ma, S. Zhang, Z. Qian, M. Xiao, J. Wu, and S. Lu. An online approach for DNN model caching and processor allocation in edge computing. *Proc of 30th International Symposium on Quality of Service* (IWQoS'22). IEEE, 2022.

[6] R. Cohen, L. Katzir, and D. Raz. An efficient approximation for the generalized assignment problem. *Information Processing Letters*, vol. 100, pp. 162 – 166, 2006.

[7] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 2094–2100, 2016.

[8] Z. Wang et al. Dueling network architectures for deep reinforcement learning. *International Conference on Machine Learning (ICML)*, pp. 1995–2003, 2016.

[9] A. Tavakoli, F. Pardo, and P. Kormushev. Action branching architectures for deep reinforcement learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*, pp. 4131–4138, 2018.

[10] M. Shahrad, R. Fonseca, I. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini. Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pp. 205–218, 2020.

[11] G. Gao, Y. Dong, R. Wang, and X. Zhou. EdgeVision: Towards collaborative video analytics on distributed edges for performance maximization. *IEEE Transactions on Multimedia*, vol. 26, pp. 9083 – 9094, 2024.

[12] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for Steiner trees. *Acta Informatica*, vol. 15, no. 2, pp. 141 –– 145, 1981.

[13] J. Li et al., Digital twin-enabled service provisioning in edge computing via continual learning. *IEEE Transactions on Mobile Computing*, vol. 23, no. 1, pp.393 – 408, 2024.

[14] J. Li, S. Guo, W. Liang, J. Wang, Q. Chen, Z. Xu, and W. Xu. AoI-aware user service satisfaction enhancement in digital twin-empowered edge computing. *IEEE/ACM Transactions on Networking*, vol. 32, no. 2, pp. 1677 – 1690, 2024.

[15] J. Li, S. Guo, W. Liang, J. Wang, Q. Chen, Z. Hong, Z. Xu, W. Xu, and B. Xiao. AoI-aware service provisioning in edge computing for digital twin network slicing requests. *IEEE Transactions on Mobile Computing*, vol. 23, no. 12, pp. 14607-14621, 2024.

[16] J. Li, S. Guo, W. Liang, J. Wu, Q. Chen, Z. Xu, W. Xu, and J. Wang. AoI-aware, digital twin-empowered IoT query services in mobile edge computing. *IEEE/ACM Transactions on Networking*, vol. 32, no. 4, pp. 3636–3650, 2024.

[17] J. Li, W. Liang, W. Xu, Z. Xu, X. Jia, A. Zomaya and S. Guo. Budget-aware user satisfaction maximization on service provisioning in mobile edge computing. *IEEE Transactions on Mobile Computing* , vol. 22, no. 12, pp. 7057 – 7069, 2023.

[18] 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 Transactions on Mobile Computing*, vol. 22, no. 5, pp.3017 – 3030, 2023.

[19] Y. Li, W. Liang, and J. Li. Profit driven service provisioning in edge computing via deep reinforcement learning. *IEEE Transactions on Network and Service Management*, vol. 19, no. 3, pp. 3006 – 3019, 2022.

[20] Y. Li, W. Liang, Z. Xu, W. Xu, and X. Jia. Budget-constrained digital twin synchronization and its application on fidelity-aware queries in edge computing. *IEEE Transactions on Mobile Computing*, vol. 24, no. 1, pp. 165 – 182, 2025.

[21] X. Liang, W. Liang, Z. Xu, Y. Zhang, and X. Jia. Multiple service model state refreshments in digital twin-empowered edge computing. *IEEE Transactions on Services Computing*, vol. 17, no. 5, pp. 2672–2686, 2024.

[22] H. Liu, S. Liu, S. Long, Q. Deng, and Z. Li. Joint optimization of model deployment for freshness-sensitive task assignment in edge intelligence. *Proc. of INFOCOM'24*, pp. 1751 – 1760, IEEE, 2024.

[23] Y. Ma et al., Profit maximization for admitting requests with network function services in distributed clouds. *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 5, pp. 1143-1157, 2019.

[24] V. Mnih et al. Human-level control through deep reinforcement learning. *Nature*, vol. 518, no. 7540, pp. 529 – 533, 2015.

[25] NetworkX. Accessed: May, 2026. [Online]. https://networkx.org/.

[26] S. Xia, Z. Yao, Y. Li, J. Pan, and L. Zuo. Personalized computation offloading and service caching for mobile edge computing in heterogeneous networks. *Proc. of Globecom'24*, IEEE, pp. 1467– 1472, 2024.

[27] J. Xie, Z. Zhou, T. Ouyang, X. Zhang, and X. Chen. Fair DNN model selection in edge AI via a cooperative game approach. *Proc. of 43rd ICDCS*, IEEE, pp. 383– 394, 2023.

[28] A. Xu, Z. Hu, X. Li, B. Chen, H. Xiao, X. Zhang, H. Zheng, X. Feng, M. Zheng, P. Zhong, and K. Li. CoMS: collaborative DNN model selection for heterogeneous edge computing systems. *IEEE Transactions on Vehicular Technology*, vol. 74, no. 2, pp. 3172–3184, 2025.

[29] Z. Xu et al., HierFedML: aggregator placement and UE assignment for hierarchical federated learning in mobile edge computing *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 1, pp. 328 – 345, 2023.

[30] Z. Xu, L. Zhao, W. Liang, O. Rana, P. Zhou, Q. Xia, W. Xu, and G. Wu. Energy-aware inference offloading for DNN-driven applications in mobile edge clouds. *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 4, pp.799 – 814, 2021.

[31] Y. Yang, Y. Shi, C. Yi, J. Cai, J. Kang, and D. Niyato. Dynamic human digital twin deployment at the edge for task execution: A two-timescale accuracy-aware online optimization. *IEEE Transactions on Mobile Computing*, vol. 23, no. 12, pp. 12262 – 12279, 2024.

[32] R. D. Yates, P. Ciblat, A. Yener, and M. Wigger. Age-optimal constrained cache updating. *Proc of IEEE International Symposium on Information Theory* (ISIT'17). IEEE, 2017.

[33] W. Zhang, D. Yang, H. Peng, W. Wu, W. Quan, H. Zhang, and X. Shen. Deep reinforcement learning based resource management for DNN inference in industrial IoT. *IEEE Transactions on Vehicular Technology*, vol. 70, no. 8, pp. 7605—7618, 2021.

[34] Y. Zhang and W. Liang. Cost-aware digital twin migration in mobile edge computing via deep reinforcement learning. *Proc. of 23rd IFIP Networking Conf.*, IEEE, 2024.

[35] Y. Zhang, L. Wang, and W. Liang. Deep reinforcement learning for mobility-aware digital twin migrations in edge computing. *IEEE Transactions on Services Computing*, vol. 18, no. 7, pp. 704-717, 2025.

[36] Y. Zhang et al., Mobility-aware service provisioning in edge computing via digital twin replica placements. *IEEE Transactions on Mobile Computing*, vol. 23, no. 12, pp. 11295-11311, 2024.

[37] Y. Zhang et al., Profit maximization of delay-sensitive, differential accuracy inference services in mobile edge computing. *IEEE Transactions on Mobile Computing*, vol. 24, no. 7, pp. 6209–6224, 2025.

[38] Y. Zhang, W. Liang, W. Xu, Z. Xu, and X. Jia. Cost minimization of digital twin placements in mobile edge computing. *ACM Transactions on Sensor Networks*, vol. 20, no. 3, pp.1 – 26, 2024.

[39] Y. Zhang, W. Liang, Z. Xu, W. Xu, and M. Chen. AoI-aware inference services in edge computing via digital twin network slicing. *IEEE Transactions on Services Computing*, vol. 17, no. 6, pp. 3154 – 3170, 2024.

[40] Y. Zhang, W. Liang, and Y. Yang. QoE-aware task executions on service models in DT-assisted edge computing. *IEEE Transactions on Mobile Computing*, vol. 25, no. 4, pp. 4585–4601, 2026.

[41] K. Zhao, Z. Zhou, X. Chen, R. Zhou, X. Zhang, S. Yu, and D. Wu. EdgeAdaptor: online configuration adaption, model selection and resource provisioning for edge DNN inference serving at scale. *IEEE Transactions on Mobile Computing*, vol. 22, no. 10, pp. 5870-5886, 2023.

[42] Z. Zhou et al., Edge intelligence: paving the last mile of artificial intelligence with edge computing. *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738 –1762, 2019.


Authorized licensed use limited to: CITY UNIV OF HONG KONG. Downloaded on November 01,2026 at 03:34:31 UTC from IEEE Xplore. Restrictions apply.
© 2026 IEEE. All rights reserved, including rights for text and data mining and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.