Computer Networks 193 (2021) 107875

Contents lists available at ScienceDirect

Computer Networks

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

Online and energy-efficient task-processing for distributed edge networks

Li Yu [a], Zongpeng Li [b,*], Jiangchuan Liu [c], Ruiting Zhou [d]

[a] School of Computer Science, Wuhan University, China
[b] Wuhan University, Wuhan, Hubei, China
[c] School of Computing Science, Simon Fraser University, Canada
[d] School of Cyber Science and Engineering, Wuhan University, China

ARTICLE INFO

Keywords:
Online learning
Internet of Things
Task offloading
Energy efficiency
Mobile edge computing

ABSTRACT

User equipment produces a series of tasks that are processed locally or remotely, falling into three categories: (i) local computing only, (ii) a fraction of the task is computed locally and the remaining task unprocessed is offloaded for remote computation, and (iii) the entire task is offloaded. Each case has attracted substantial attention in recent studies, where a delay-constrained non-linear optimization problem is often formulated. The solutions employed are either based on Lagrange duality, heuristic search, or dynamic programming. To our knowledge, there is no unifying task-processing orchestrator that is an online tailored solver for learning the model-free problems, encapsulating the three cases above. We fill this gap and present the first attempt on an innovative actor-critic reinforcement learning approach in consideration of the energy-efficiency, to compute the asymptotically optimal solutions via decomposing the comprehensive optimization into sub-problems. Rigorous theoretical analyses and experience-driven simulations demonstrate significant advantages over the benchmark approaches, in terms of task-processing delay, power efficiency, and convergence time.

1. Introduction

With the prevalence of next-generation wireless systems, the wireless world is to be interconnected without barriers such as the Internet of Things (IoT), Vehicle to Internet, and Vehicle to Grid. Correspondingly, intelligent devices are continuing to produce a plethora of services termed as *tasks*, such as online gaming, video streaming, etc. The exploding growth of tasks is now on a way to exhaust the capabilities and energy of IoT devices, resulting in the degradation of the quality of experience on user's side, especially in ultra-dense networks.

Fortunately, Mobile Edge Computing (MEC), as a promising paradigm of offloading computation for users, can boost their computation capability and shorten the operational time of low-power wireless devices. IoT devices can offload part or all of their computation-intensive tasks to the resource-rich MEC servers in proximity to shorten their computing time [1].

A fine-grained study was made concerning the task-processing types of the partial or binary offloading, the scheme of offloading computation, the main metrics of model, and the task duration for task completion as sketched in Table 1, which shows that the existing works [2,3] operate in an offline fashion (i.e., the task workload is known in advance) and hence cannot benefit certain IoT users, such as crowdsensing with uncertain sensed data arrival, and real-time environment predication.

A flurry of latest research activities on task offloading were dedicated to gaining insights from reinforcement learning, where an agent learns to take actions to yield the most payoff or minimal cost via systematic (or environmental) interaction. Different from the supervised learning which learns from the samples provided by an experienced supervisor, reinforcement learning has to accumulate experience though exploration and exploitation, while encountering various uncertainties about the surroundings. The procedure is called trial-and-error learning with delayed reward (penalty) [11,12]. The agent stores the accumulated experience via experience replay, gaining the prior knowledge. As a consequence, a learning-enabled strategy exhibits potential for tackling the computation-intensive and delay-sensitive services in IoT networks.

Furthermore, it is observed that i) the existing researches on task-offloading computation are mainly classified into three categories: partial task offloaded to servers [2,4] and binary offloading computation (i.e., the admitted traffic is either processed by local device or totally offloaded to MEC servers) [5,6], and the combination of partial and binary computation [7,8]. However, the formulated models of task offloading are diverse, such as time model and utility model, which are mostly based on models to address the corresponding problems. For example, game-theoretic methods and heuristic searching algorithms, which process tasks in an offline manner; ii) moreover, the problems

* Corresponding author.
E-mail address: zongpeng@whu.edu.cn (Z. Li).

https://doi.org/10.1016/j.comnet.2021.107875
Received 25 April 2020; Received in revised form 2 January 2021; Accepted 27 January 2021
Available online 24 February 2021

Table 1
Comparison of several task-processing approaches.

References	Partial or binary offloading	Objective of optimization	Method of solution	Task duration	Energy consumption
[2]	Partial	Delay	Iterative heuristic algorithm	Low	Medium
[4]		Networking cost	Packet assignment algorithm	N/A	N/A
[5]	Binary	Computation rate	Bi-section search algorithm	Medium	Medium
[6]		Delay	Heuristic search		N/A
[7]	Both	Computation rate	Optimization without learning	Medium	Medium
[8]		Energy efficiency	No learning		Low
[9]	N/A	Cost efficiency	Iterative algorithm	N/A	N/A
[3]		Cost efficiency	Dynamic programming		N/A
[10]		Energy efficiency	Heuristic search algorithm		Low
[11]	Partial	Delay	ACRL	Low	N/A
[12]					
OTS	Both	Energy efficiency & delay	Online tailored solver with energy efficiency	Low	Low

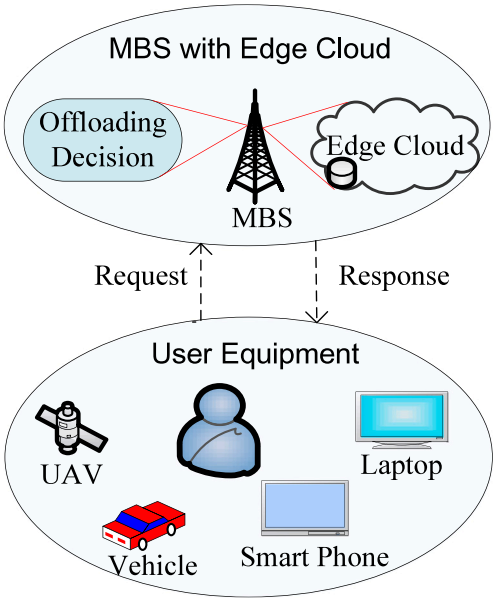

Fig. 1. System architecture.

lack of the precise model to formulate will impede the derivation of optimal solution. For instance, it is hard to formulate the real-time and accurate model of the communication capacity with Signal to Interference and Noise Ratio (SINR), because of the existence of interference in the dynamic network environments; iii) the energy-efficiency formulation of IoT devices is hardly covered via using the improved actor-critic reinforcement learning algorithm. To our knowledge, there have been no studies about a unifying task-processing orchestrator that is an online tailored solver for learning the model-free model in consideration of the cases of the partial and binary task offloading.

Motivated by the above observations, we have made the following contributions.

- A unifying task-processing orchestrator is first explored, which is an online tailored solver taken the cases of the partial and binary task offloading into account for learning the model-free problems. To our knowledge, the approach is the first work.
- The energy-efficiency model of IoT devices is elaborately devised in a distributed manner according to the real-world settings, shifting the conventional architecture research to the user-oriented IoT devices.
- The first attempt on an innovative actor-critic reinforcement learning approach considered the energy efficiency is made, which derives an asymptotically optimal solution.
- Extensive simulation analysis demonstrates our proposed approach outperforms the state-of-the-art in terms of task-processing delay, energy efficiency and convergence time.

The remainder of this paper is organized as follows. Section 2 reviews the related work. Preliminaries and problem formulation are elaborated on in Section 3. Our decomposition method is subsequently presented in Section 4. In Section 5, the in-depth analysis on the result of simulation is described in detail. Finally, we conclude the paper in Section 6.

2. Related work

As a promising technology to augment resource usage of device and conserve its energy, offloading tasks to MEC server has been attached great concern [2,10] and [13]. Park et al. [4] proposed a game-based multi-user computation offloading algorithm. Geng et al. [10] investigated to minimize energy consumption formulated as a mixed-integer nonlinear programming problem and exploited the heuristic algorithm to solve the binary offloading decision and task scheduling problems. The mixed-integer non-linear programming concerning task offloading in the unmanned aerial vehicle networks was explored [7]. Bai et al. [8] leveraged a heuristic offloading decision algorithm. Although these works [7] and [8] consider the cases of both partial and binary offloading computing, some problems are not taken into account. For example, the specific number of a task for offloading is not considered as well as whether the task is more applicable to be offloaded or not.

Yu et al. [1] devised a task-processing time model for managing the task-offloading assignment. Nevertheless, the designed variable was binary rather than the specific ratio for allocation that has proved to be more significant to optimize the task-processing delay [11,12] and [14]. Therefore, it is much-desired to design a unifying orchestrator that can manage all circumstances covering the binary offloading, and the partial offloading that involves how much percentage of the admitted task can be offloaded to the MEC server.

Xu et al. [15] considered joint service caching and task offloading using the Lyapunov optimization and Gibbs sampling to achieve the provable close-to-optimal performance in terms of computation latency. Zhou et al. [16] proposed a deep-learning-based approach to predict a congestion event in advance so that the communication ratio by uplink (or by downlink) can be flexibly adjusted, circumventing potential congestion in a proactive manner. Liu et al. [17] designed an actor-critic learning algorithm to conduct the UAV control. However, different from them, we focus on the allocation of offloading computing. Xiao and Krunz [18] emphasized energy efficiency in fog computing in Tactile Internet. Unlike them, we shift the focus on energy efficiency from the network-wide fog node to system-wide end-users.

Subsequently, we will exhibit an innovative and comprehensive formulation of our task-offloading time model.

Table 2
Some notations.

Notation	Description	Notation	Description
e^{max}	Maximum capacity of finite-size battery	w_n	Fairness at time slot n
β	Discount factor and its range is $[0, 1)$	$\boldsymbol{D}$	Size of dataset in replay
λ	Parameter of controlling update rate	$\mathbb{U}$	Set of users within one BS
c_u	Computing capacity of device for user u (cycles per unit time)	e_{loss}	Energy consumption at unit time
$e_{com}(n)$	Self-power loss at time slot n	e_{thre}	Threshold of energy in device
$e_{har}(n)$	Amount of energy harvested at time slot t	ϵ	Greedy selection probability
$e(n)$	Energy stored in the battery at the end of slot $n-1$ $(n \in \mathbb{N})$	$\mathbb{N}$	Set of epoches
$e_{ser}(b, n)$	Power loss of performing a task u	ξ^{π} and ξ^{Q}	Hyper-parameters
$\pi^{'}(.)$	Corresponding target network with parameters $\theta^{\pi^{'}}$	$\pi(.)$	Actor network with parameters θ^{π}
$Q^{'}(.)$	Corresponding target network with parameters $\theta^{Q^{'}}$	$Q(.)$	Critic network with parameters θ^{Q}
$t^{d}_{u,b}(n)$	Deadline of performed task b for user u at n	D_u	Number of data samples of user u

3. Preliminaries and problem formulation

In this section, some related preliminaries and problem formulation are described. For ease of reference, the major notations are summarized in Table 2.

3.1. Actor-critic reinforcement learning

Reinforcement learning, in general, is composed of an agent interacting with the learning objective (i.e., environment) in the discrete decision epochs. The agent (i.e., the device or cellular system) observes the environmental state s_n and takes an appropriate action a_n under certain policy $\pi(s)$ at each decision epoch n. It receives an immediate reward of R in an iterative manner. The overarching objective of agent is to find an optimal policy $\pi^*(s)$ mapping a state to a deterministic action or a probability distribution over the stochastic actions space according to the maximal value of cumulative discounted reward under the current state–action pair, which is expressed by

$$Q^*(s_n, a_n) = R(s_n, a_n) + \beta(\sum_{s'_n} P(s_n, a_n, s'_n) \max_{a'_n} Q^*(s'_n, a'_n)) \tag{1}$$

where $Q^*(s_n, a_n)$ represents the Q-value of the state–action pair (s_n, a_n) under an optimal policy. $P(s_n, a_n, s'_n)$ denotes agents' transition possibility from current state s_n to next state s'_n for given action a_n.

The procedure of Actor-Critic Reinforcement Learning (ACRL) adopted a neural network (or even the deep convolutional neural network abbreviated as DNN) to approximate the Q-value function, which involves the actor part and the critic one. The actor part intends to search for an optimal or suboptimal strategy that is parameterized. Then it generates actions according to the observed environmental state. Whereas the critic one aims to estimate and criticize the current policy by receiving rewards, which, therefore, is called the estimated Q-function. The evaluated Q-function is guided by the Temporal Difference (TD) error and trains the corresponding critic networks. The TD error is used to reconcile the gap between the actor part and the critic one, mostly diminishing the gap. The actor networks then utilize the output of the critic network to update its parameters of the policy [19].

Subsequently, the endeavors in DeepMind corporation introduced the DDPG approach [20]. Vamvoudakis et al. [21] used the method for continuous control. The core idea of DDPG was to coordinate an actor function $\pi(s_n|\theta^{\pi})$ about the parameter θ^{π} with a critic function $Q(s_n, a_n|\theta^{Q})$ with respect to the parameter θ^{Q}. The parameterized actor function returns a Q-value. The parameterized critic function then criticizes the Q-value how good it is. The parameters θ^{π} of the actor networks with the actor function $\pi(n)$ can be updated by applying the chain rule to the expected cumulative rewards, both of which are respectively given by [22]

$$\pi(n) = R(s_n) + \beta\pi(s'_n|\theta^{\pi}) \tag{2}$$

$$\begin{aligned} &\nabla_{\theta^{\pi}} J = \mathbb{E}[\nabla_{\theta^{\pi}} \log \pi_{\theta}(a|s) Q(s, a|\theta^{Q})|_{s=s_n, a=\pi(s_n|\theta^{\pi})}] \\ &\approx \mathbb{E}[\nabla_{\theta^{\pi}} Q(s, a|\theta^{Q})|_{s=s_n, a=\pi(s_n|\theta^{\pi})}] \\ &\approx \mathbb{E}[\nabla_{a} Q(s, a|\theta^{Q})|_{s=s_n, a=\pi[s_n]} \cdot \nabla_{\theta^{\pi}} \pi(s|\theta^{\pi})|_{s=s_n}] \end{aligned} \tag{3}$$

where J represents the expected cumulative reward. π is the state value concerning θ^{π}. $\mathbb{E}[.]$ is the estimated value. Correspondingly, the critic function in the training networks is supervised by the loss function [23] that is denoted as the TD error $\delta_{\theta^Q}(n)$. It is calculated by

$$\delta_{\theta^Q}(n) = \mathbb{E}[(Q'(s'_n, a'_n|\theta^{Q}) - Q(s_n, a_n|\theta^{Q}))^2] \tag{4}$$

where $Q'(s'_n, a'_n|\theta^{Q})$ is the estimated target value which is expressed by

$$Q'(s'_n, a'_n|\theta^{Q}) = R(s'_n, a'_n) + \beta Q(s'_n, \pi(s'_n|\theta^{\pi})|\theta^{Q}) \tag{5}$$

3.2. System architecture

We consider there is a set $\mathbb{U}$ of user equipment (termed as users) with power-limited IoT devices (e.g., vehicle, mobile phone, UAV, and so on) as shown in Fig. 1. Each user is denoted as u, $u \in \mathbb{U}$, $\mathbb{U} = [1, 2, \ldots, U]$. Users randomly generate a variety of services denoted by a set $\mathbb{B}$. Each service is denoted as a task b, $b \in \mathbb{B}$, $\mathbb{B} = [1, 2, \ldots, B]$. The task consists of different factors for different objects. For example, the user requests the service. The task is mainly constructed by data for task offloading. While the edge will be processing the data sent by users. The implement of the procedure of computing and assigning resources in the edge server will contain the executable code and the data. The code may be Java code, virtual machine or container etc. In our manuscript the executable code is generated under the premise that the algorithms are compiled. Furthermore, the algorithm is related to the iterations of the deep actor-critic networks. That is, the dynamic learning procedure has relevance to time slots, the number of users, CPU frequency and workload, and so forth, which will be further illustrated in Algorithm 1 in Section 4. The codes are embedded in the edge server. Its related parameters of the algorithm are debugged via extensive simulations in Section 5. Consider that these users can communicate with the nearest Macro Base Station (MBS) equipped with an edge cloud server. The edge is responsible for the user's task offloading to alleviate the user's workload, which is our major concern. As shown in Fig. 1, users can request the service for task offloading. The MBS makes the offloading decisions through the online tailored algorithm with energy efficiency that is illustrated in Section 4, and responses to users. Correspondingly, users estimate their energy by leveraging the power evolution algorithm, sending messages concerning their energy state, and task-offloading information when they are connected to the MBS via wireless communication. Due to the task's property, i.e., the encrypted file to be processed and the video containing privacy and so on, these tasks should not be offloaded to the MEC servers. Nevertheless, whether a task can be offloaded or not are illustrated in detail in [24], which is beyond our scope. We focus on the task-processing time for the offloadable and non-offloadable task, and on how to assign the offloading percentage for the offloadable task in dynamics.

We consider a finite time horizon with duration $\mathbb{N}$ that is discretized into N equal time slots. It is denoted by $n \in \mathbb{N}$, $\mathbb{N} = [1, 2, \ldots, N]$. Assume a quasi-static scenario where the occupying wireless channels of task b is stable in a time slot that may last a few seconds, while the temporal

and geographical variations probably incur across different time slots due to the bursty nature of human-generated traffic. Task requested by the user can be processed by the device itself if it has adequate energy left and enough computing resources (e.g., CPU, memory, etc.). If the task can be offloaded to the edge server, it can be processed by the alternative edge computing in proximity. Since the available computing capacity (in cycles/second) is determined by the stabilized working frequency of CPU and the current CPU workload (i.e., the percentage of the processing capacity occupied by other users) in a time slot [25], the MBS with edge server decides the allocation of channel communication capacity according to the corresponding algorithms that return the provisioning of power and task-offloading percentage. In other words, the edge decides the proportion of task offloading for different users based on its network and user's states involving the channel capacity and available memory and user's energy, etc. Correspondingly, the user performs the specific ratio for tasks offloaded to the edge.

Generally, the MEC server is equipped with high-performance traffic processing units and has a rich resource pool. Therefore, the queuing time of the offloaded traffic is much smaller than the transmission time of workload [18]. For simplification, assume that the offloadable task can be directly dealt with and fed backed by the nearest MEC server without the task-forwarding operation between servers since the MEC server is resource-rich and computation-powered.

3.3. Energy model

In order to shed meaningful insights into our design of the algorithm, a tractable energy model is developed as follows. Each user has an energy harvesting circuit and a size-constrained battery that can store energy for its operations.

Unlike [9] and [13], which explored the optimization objective of economical energy efficiency, we consider a more generic scenario where IoT devices are equipped with renewable energy by conventional power grid or new rechargeable energy (e.g., solar, radio signal, and so on). The finite-size battery can be viewed as a backlog in an energy queue. Provided that $l(b,n)$ (in cycles) stands for the number of cycles performed by CPU for task b at time slot n, and it can be acquired by the CPU monitor in the device. Obviously, the executing time of task can be calculated by $\frac{l(b,n)}{c_u}$. The residual energy $e_{left}(b,n)$ of processing the task b at slot n can be estimated by

$$e_{left}(b,n) = e(n-1) + e_{har}(n) - e_{com}(n) - e_{ser}(b,n) \tag{6}$$

where $e_{ser}(b,n) = (\frac{l(b,n)}{c_u}) * e_{loss}$.

After servicing all tasks, the energy left of the device will follow the constraint of utilizable energy below.

$$e_{left}(b,n) \geq e_{thre}(n) \tag{7}$$

The queue of energy backlog $e(n)$ evolves as follows when the constraint of Eq. (7) is satisfied in each slot.

$$e(b,n+1) = \min\{e_{left}(b,n), e^{max}\},\ \forall b \in \mathbb{B}, \forall n \in \mathbb{N}. \tag{8}$$

Afterwards, we will elaborate on the procedure of formulating the problem.

3.4. Problem formulation

Based on the aforementioned system and energy model, we focus on the task-processing time for the offloadable task and how to assign the offloading percentage of the offloadable task. Herein, we assume the generated tasks are independent and identically distributed. As the system model illustrated in Section 3.2, the delay model will exclude the forwarding and queuing time. Task-processing computation can be discussed in different cases as follows.

Binary task-offloading cases: Binary task-offloading cases incorporate that a task is either processed locally or it is completely offloaded to MEC server as follows.

Case 1-1: Local task-processing case

Assume that a new task b produced by user u at time slot n can be recorded by packet message, including the size of task b denoted as f_b (in bits). CPU in the device performs the request by transforming f_b into the number of CPU cycles required for computing the equivalent task b. Let $\aleph_b$ denote the executed cycles of CPU for the equivalent task. IoT device has a constant computing capacity c_u (in cycles/second) [6,22]. The delay for processing a task locally can be calculated by

$$t_{loc} = \frac{\aleph_b}{c_u} \tag{9}$$

Case 1-2: Total task-offloading case

According to the system model elaborated on in Section 3.2, the task-processing time involves two parts in the situation where a requested task is totally offloading to the MEC server. One part is the round-trip transmission latency between user and edge, which is denoted as $2t_{del}$. The other is the task-processing time in the MBS, which is denoted as t_{mec}. The expression is

$$t_{rem} = t_{mec} + 2t_{del} = \frac{\aleph_b}{c_m} + 2\frac{f_b}{p^n_{u,b}} \tag{10}$$

where c_m represents the average computing capacity (i.e., the average number of CPU cycles to process the unit-bit task in unit time), which is a constant determined by the MEC server's hardware settings [9] and [12]. $p^n_{u,b}$ stands for the channel communication capacity (in bits/second) for task b of user u at time slot n, which is variable. Note that its exact expression is unknown due to the existence of channel interference among the different tasks for offloading computing.

Partial task-offloading case: The requested task b is partially processed locally, and the remaining part is offloaded to MEC.

Case 2: Partial task-offloading case

The provisioning algorithm of explicitly offloading proportion is designed to reduce the task-processing time which is a crucial criterion for time-sensitive user equipment. The partial task-offloading delay for a task is given by

$$\begin{aligned} t_{par} &= t_{loc} + t_{mec} + 2t_{del} \\ &= \frac{\aleph_b}{c_u} * x^n_{u,b} + \frac{\aleph'_b}{c_m} * (1 - x^n_{u,b}) + 2\frac{f'_b}{p^n_{u,b}} * (1 - x^n_{u,b}) \end{aligned} \tag{11}$$

where $x^n_{u,b}$ represents the task-offloading ratio for task b of user u at slot n, and $\aleph'_b$ is the required number of cycles of CPU for the offloaded part of task b. Likewise, f'_b denotes the needed number of the offloaded bits to MEC. That is, it is the remaining size of the task which is processed by MEC.

Note that the aforementioned three cases merely consider the task-processing delay for each case separately. Later on, a unifying orchestrator that encapsulates the cases of the binary and partial task-offloading computing will be elaborated on.

A variable $y^n_{u,b}(y^n_{u,b} \in [0,1])$ is introduced, representing whether a task is offloaded binarily or partially. That is, the total task is processed locally if $y^n_{u,b} = 0$. The total task is offloaded to server if $y^n_{u,b} = 1$. Otherwise, the task is the partially offloading case. Herein, for ease of description, let $\frac{\aleph_b}{c_u} = \mu_1$, $\frac{\aleph_b}{c_m} = \mu_2$ and $\frac{\aleph'_b}{c_m} = \mu_3$. The total task-processing time of all tasks generated by users is t_{all}, which is formulated as

$$t_{all} = \begin{cases} \sum_{u=1}^{U}\sum_{b=1}^{B} \mu_1, & if \quad y^n_{u,b} = 0; \\ \sum_{u=1}^{U}\sum_{b=1}^{B} (\mu_2 + 2\frac{f_b}{p^n_{u,b}}), & if \quad y^n_{u,b} = 1; \\ \sum_{u=1}^{U}\sum_{b=1}^{B} (\mu_1 x^n_{u,b} + \mu_3(1 - x^n_{u,b}) + 2\frac{f'_b}{p^n_{u,b}}(1 - x^n_{u,b})), & \\ & if \quad 0 < y^n_{u,b} < 1. \end{cases} \tag{12}$$

where there exist three variables, i.e., $x^n_{u,b}$, $y^n_{u,b}$ and $p^n_{u,b}$. Different from the label of $y^n_{u,b}$, $x^n_{u,b}$ stands for the specific task-offloading proportion. The value of $y^n_{u,b}$ relies on the property of user's task, which can be obtained from the interaction with users. The edge is responsible for

the task-offloading and the allocation of communication capacity under the precondition of the model-free surroundings.

The total task-processing time t_{all} is expected to be as minimal as it is, enhancing the quality of user's experience. The total time model is abstracted as a comprehensive optimization problem **P1** with respect to $x^n_{u,b}$, $y^n_{u,b}$ and $p^n_{u,b}$ at time slot n.

$$\mathbf{P1}: \min_{\mathbb{Y},\mathbb{X},\mathbb{P}} \ t_{all}(x^n_{u,b}, y^n_{u,b}, p^n_{u,b}) \quad \forall u \in \mathbb{U}, \forall b \in \mathbb{B}, \forall n \in \mathbb{N} \tag{13}$$

$$\begin{aligned} s.t.\ & C1 \quad x^n_{u,b} \in [0,1] \\ & C2 \quad y^n_{u,b} \in [0,1] \\ & C3 \quad \sum_{u=1}^{U} \sum_{b=1}^{B} p^n_{u,b} \le P^{max}(n) \\ & C4 \quad p^n_{u,b} \ge 0 \\ & C5 \quad e_{left}(b,n) \ge e_{thre}(n) \\ & C6 \quad t_{u,b}(n) \le t^d_{u,b}(n) \end{aligned}$$

where the variables of $\mathbb{X}$, $\mathbb{Y}$ and $\mathbb{P}$ represent the corresponding set respectively, i.e., $\mathbb{X} = [x^n_{u,b}]$, $\mathbb{Y} = [y^n_{u,b}]$ and $\mathbb{P} = [p^n_{u,b}]$. The inequation $C3$ represents each request b is designated to a dedicated communication capacity (in bits) for task offloading. The overall capacity of all tasks generated by users is less than the maximal fronthaul capacity $P^{max}(n)$ of server in practice at the slot n. The notation of $P^{max}(n)$ is referred to as the maximally total downlink transmitting capacity that edge server can support [9]. The inequation $C5$ stands for the constraint of energy.

The constraint of inequation $C6$ holds since the processed result of offloading, transmitting and computing should be within a specific deadline that is an expired time to response to user u. Note that the constraint $C6$ can efficiently avoid potential safety hazards and low-efficiency tasks to be processed (e.g., the task with occupying the shared resource for long time, and the task waiting for entering the encrypted data, etc.).

4. Decomposition method

In terms of the above optimization problem **P1**, there have been many conventional approaches to solve such as the heuristic search methods [2,5,6,10] (e.g., ant colony search [2], simulated annealing algorithm [10] etc.), Lagrange duality [1] and non-linear or linear programming [3,7,8], and so forth. Nevertheless, these schemes just fitted the low-dimension searching space in an offline fashion, which are inapplicable to the real-time scenarios (e.g., newly created task to be allocated via the exact percentage of task offloading), and to the unknown surroundings with multiple dimensions such as the interference among the different users, the time-varying energy left for users and so on. To resolve **P1**, we notice that the assignment of offloading percentage is no longer to perform when $y^n_{u,b} = 0$, and that all tasks are offloading to the edge with $x^n_{u,b} = 1$ when $y^n_{u,b} = 1$. Based on the coupling relationship, **P1** can be decomposed into the resulting three sub-problems that are corresponding to the different approaches to address.

4.1. Equivalent subproblems

Note that the value of $y^n_{u,b}$ depends on the service's property (i.e., the non-encrypted file will be labeled as $y^n_{u,b} \in (0,1]$, which can be easily obtained in the field of the header of each task.)

Case 1-1: Local Task-Processing Case

Assuming that the task b of user u is totally be processed by the device itself. Namely $y^n_{u,b} = 0$. The **P1** is transformed into the following **P1-1**.

$$\begin{aligned} \mathbf{P1\text{-}1}: \quad & \min \quad t_{all} = t_{loc} = \sum_{b=1}^{B} \frac{\aleph_b}{c_u} = \mu_1 \\ & s.t. \quad C7: \quad y^n_{u,b} = 0, \quad \forall u \in \mathbb{U}, b \in \mathbb{B}, n \in \mathbb{N} \end{aligned} \tag{14}$$

Case 1-2: Total Task-Offloading Case

Consider that the task b of user u is totally be processed by MEC. Namely $y^n_{u,b} = 1$. The **P1** is transformed into the following **P1-2**.

$$\begin{aligned} \mathbf{P1\text{-}2}: \quad & \min_{\mathbb{P}} \quad t_{all}(p^n_{u,b}) \\ & s.t. \quad C3 \quad \& \quad C4 \quad \& \quad C5 \quad \& \quad C6 \\ & C8: \quad y^n_{u,b} = 1, \quad \forall u \in \mathbb{U}, b \in \mathbb{B}, n \in \mathbb{N} \end{aligned} \tag{15}$$

Observe that the totally task-processing time $t_{all}(\cdot)$ is a function with respect to the one-dimensional variable of the communication capacity $p^n_{u,b}$. Since a myriad of users compete for the communication capacity under a shared wireless channels, without the loss of generality, there exists the internal interference due to the environmental noise and inherent interference in the same spectrum band [26]. The expression of $p^n_{u,b}$ is more complex than the simple Shannon equation. Concretely, $p^n_{u,b}$ is an unknown function, which is as opposed to the well-defined model in the circumstance of interference [27].

Case 2: Partial Task-Offloading Case

As for the partial offloading computation, the problem **P1-3** is formulated as

$$\begin{aligned} \mathbf{P1\text{-}3}: \quad & \min_{\mathbb{X},\mathbb{P}} \quad t_{all}(x^n_{u,b}, p^n_{u,b}) \\ & s.t. \quad C1 \quad \& \quad C3 \quad \& \quad C4 \quad \& \quad C5 \quad \& \quad C6 \\ & C9: \quad y^n_{u,b} \in (0,1), \quad \forall u \in \mathbb{U}, b \in \mathbb{B}, n \in \mathbb{N} \end{aligned} \tag{16}$$

Observe that the total task-processing time $t_{all}(\cdot)$ is a function regarding the two-dimensional variables of task-offloading percentage and communication capacity.

Different from the work [1], we focus on the actor-critic deep reinforcement learning algorithm with energy efficiency to address the problems **P1** with the continuous space. (See [28]).

4.2. Online tailored solver

As aforementioned, we are mainly to solve the problems of **P1-2** and **P1-3** in the cases of the binary and partial task processing respectively. To kill two birds with one stone, we normalize the variable $p^n_{u,b}$ to the range of $[0,1]$ whose range is the same to $x^n_{u,b}$. Aiming to cope with the tricky variables with featuring continuous rather than binary control variables that can be tackled by the conventional reinforcement learning (i.e., Deep Q-learning Network (DQN) [1,11,12], and [29]), an online tailored solver in consideration of the partial and binary task offloading is elaborately developed to address the model-free problem.

The basic components of agent (device) via defining the Markov Decision Process (MDP) [30] is outlined before modeling the learning procedure.

Definition 1 (*MDP*).: A MDP is a tuple encompassing $\{S, A, R, P\}$, where S is state space; A is action space of agent; $R: S \times A \to R$ is a reward function mapping state–action pairs to rewards; $P: S \times A \times S \to [0,1]$ is transition function.

The agent decides how much percentage of the offloading task for each user is to be assigned. It hinges on the task-processing cases and its performed energy consumption. Therefore, the state $s^n_{u,b}$ of user's task b at decision epoch n is defined as $s^n_{u,b}(t^d_{u,b}, y^n_{u,b}, e_{left}(b,n))$. The state space of device is $S = [s^n_{u,b}], \forall u \in \mathbb{U}, b \in \mathbb{B}, n \in \mathbb{N}$.

The action of agent is to perform an appropriate assignment that specifies how much ratio of task offloading and communication capacity is to be assigned to the corresponding unknown variables in each slot. Therefore, the action for task b of user u at the slot n is $a^n_{u,b}$, $a^n_{u,b} \in A$, $A = [\mathbb{X}, \mathbb{P}]$, where $\mathbb{X} = [x^n_{u,b}]$ and $\mathbb{P} = [p^n_{u,b}]$ for $\forall u \in \mathbb{U}, b \in \mathbb{B}, n \in \mathbb{N}$. Further, $x^n_{u,b} = \frac{m_{u,b}}{m_u}$, $\forall b \in \mathbb{B}$, where $m_{u,b}$ represents the number of the offloaded task b at decision epoch n, and m_u is the total number of the offloaded tasks for user u in a time slot. Clearly, $x^n_{u,b} \in [0,1]$. Similarly, $p^n_{u,b} = \frac{q_{u,b}}{q_u}$, $\forall b \in \mathbb{B}$, where $q_{u,b}$ denotes the amount of capacity for task

Algorithm 1: Online Tailored Solver with Energy Efficiency.

Input: Initialize $\pi(\cdot)$ with θ^{π}, and $Q(\cdot)$ with θ^{Q};
Initialize $\pi'(\cdot)$ with $\theta^{\pi'} \doteq \theta^{\pi}$, and $Q'(\cdot)$ with $\theta^{Q'} \doteq \theta^{Q}$;
Initialize S, A, β and D.
Output: A.
1: **for** $n = 1$ to N **do**
2: Receive the initial observed state $s^n_{u,b}$
3: $a^n_{u,b} \leftarrow select(A)$ via $\epsilon-$greedy algorithm [28]
4: Perform $a^n_{u,b}$ and observe the immediate reward $R^n_{u,b}$ and next state $s^{n+1}_{u,b}$
5: **if** (Constraint $C6$ holds && $y^n_{u,b} \in (0,1]$) **then**
6: Put the transition sample $(s^n_{u,b}, a^n_{u,b}, R^n_{u,b}, s^{n+1}_{u,b})$ into dataset D
7: **while** $D \neq \varnothing$ **do**
8: Sample minibatch of D_u samples from D
9: Calculate the targeted value $\pi'(n)$ and $Q'(n)$:
$\pi'(s'_n|\theta^{\pi}) = R(s'_n) + \beta\pi(s'_n|\theta^{\pi})$
$Q'(s'_n, a'_n|\theta^{Q}) = R(s'_n, a'_n) + \beta Q(s'_n, \pi(s'_n|\theta^{\pi})|\theta^{Q})$
10: Calculate gradients of the actor and critic networks by Eq. (3), respectively: $\nabla_{\theta^{\pi}}\pi$ and $\nabla_{\theta^{Q}}Q$
11: Calculate TD-error $\delta_{\theta^{\pi}}$ and $\delta_{\theta^{Q}}$:
$\delta_{\theta^{\pi}}(n) = E[(\pi'(s'_n|\theta^{\pi}) - \pi(s_n\theta^{\pi}))^2]$
$\delta_{\theta^{Q}}(n) = E[(Q'(s'_n, a'_n|\theta^{Q}) - Q(s_n, a_n|\theta^{Q}))^2]$
12: Update the parameters of the actor and critic networks, respectively:
$\theta^{\pi}(n+1) \leftarrow \theta^{\pi}(n) + \xi^{\pi} \cdot \delta_{\theta^{\pi}}(n) \cdot \nabla_{\theta^{\pi}}\pi$; reset $\nabla_{\theta^{\pi}}\pi \leftarrow 0$
$\theta^{Q}(n+1) \leftarrow \theta^{Q}(n) + \xi^{Q} \cdot \delta_{\theta^{Q}}(n) \cdot \nabla_{\theta^{Q}}Q$; reset $\nabla_{\theta^{Q}}Q \leftarrow 0$
13: Update the parameters of the corresponding target networks:
$\theta^{\pi'}(n+1) \leftarrow \lambda\theta^{\pi}(n) + (1-\lambda)\pi'(n)$
$\theta^{Q'}(n+1) \leftarrow \lambda\theta^{Q}(n) + (1-\lambda)Q'(n)$
14: **end while**
15: **else**
16: $R^n_{u,b} \leftarrow 0$; $a^n_{u,b} \leftarrow 0$
17: **end if**
18: Send the push-based services
19: **end for**

b at epoch n, and the total channel capacity for the offloaded tasks of user u in a time slot is denoted as q_u. In addition, $p^n_{u,b} \in [0,1]$.

Definition 2 (*Fairness*)**.** The task-offloading fairness for users, denoted as η^n, is referred to as the maximum fairness of task offloading for the B tasks generated by U users at slot n, which is given by

$$\eta^n = \| \frac{\overline{a^n_{u,b}}}{\sum_{b\in B} a^n_{u,b}} - \frac{D^{ave}_b}{\sum_{b\in B} D^{ave}_b}, \forall u \in U, b \in B \|_{\infty} \tag{17}$$

where $\overline{a^n_{u,b}} = \lim_{N\to\infty} \frac{1}{N} \sum_{\tau=0}^{N-1} \mathbb{E}[a^{\tau}_{u,b}]$ denotes the average of time over N slots for a random process $a^n_{u,b}$. $\| \cdot \|_{\infty}$ stands for $\mathcal{L}_{\infty}$ norm. The first term at the right of Eq. (17) is the ratio of $a^n_{u,b}$ between the time-average of the allocation of task-offloading proportion for task b and the overall time of allocation for B tasks, and the second term represents the task-offloading ratio of task b between the average of generated dataset for task b and the overall dataset of edge.

The rewards is devised as

$$R^n_{u,b} = \frac{\eta^n * \sum_{b=1}^{B}(a^n_b - a^{n-1}_b)}{\sum_{b=1}^{B}[e(b,n) - e(b,n-1)]} \tag{18}$$

where $(a^n_b - a^{n-1}_b)$ is the marginal gain (an incremental allocation ratio). $(e(b,n) - e(b,n-1))$ is the incremental cost concerning energy consumption. $\eta^n * (a^n_b - a^{n-1}_b)$ can be regarded as an effective incremental assignment by adding a discount to the actual incremental value if the increment induces unfairness. Overall, the reward can be considered as energy efficiency. The agent aims to find a policy, making the cumulative reward maximal. It is equivalent to maximize the average of energy efficiency.

Algorithm 2: Power Evolution Algorithm for UE.

Input: Initialize D_u and A;
Output: $s^n_{u,b}$.
1: **for** $n \leftarrow 1$ to N **do**
2: **if** (receiving push-based service) **then**
3: Calculate $e_{left}(b,n)$ according to Eq. (6)
4: **else**
5: **if** (Eq. (7) holds) **then**
6: $y^n_{u,b} \leftarrow$ rand (0,1)
7: Generate $t^d_{u,b}$ of task offloading
8: Update energy $e(b,n+1)$ according to Eq. (8)
9: Send user state $s^n_{u,b}$ to MBS
10: **end if**
11: **end if**
12: **end for**

The Online Tailored Solver with energy efficiency (OTS) is portrayed in Algorithm 1. The general framework of the actor-critic reinforcement learning was employed [11] and [12]. However, different from them, we emphasize that the constraints of the task-performing time should be satisfied. If not, the service will not be performed. The reward to supervise the selection of the next action is set to 0, which is the most severe penalty (line 17). Because, according to our extensive study (i.e., [2,6,11] and [12]), it can hardly make sense to process a task in an expired time if the task has a completed deadline. Note that the reward $R^n_{u,b}$ is calculated by Eq. (8), Eq. (17) and (23), which embodies the fairness of task-offloading assignment for all tasks. Moreover, the reward takes energy and task-processing deadline into account, customizing the requirements of different users (from line 8 to line 15). Meanwhile, the power evolution of users is illustrated in Algorithm 2. We consider that the tasks are offloadable to the edge to alleviate the user's computing burden, thereby generating a random number via the function of rand (line 6). Each user calculates and updates its energy according to the energy model described in Section 3.3, and communicates with the edge in MBS through sending the user's state (line 1 to 10).

4.3. Theoretical analysis

In the procedure of actor-critic learning, the stochastic gradient descent method [31,32] amd [33] is adopted, minimizing the loss functions. In order to analyze the rationale of actor-critic learning in Algorithm 1, we further make the theoretical illustration.

The mechanism of ACRL has two groups of networks, the original actor-critic networks and the target actor-critic networks. For ease of notations, let W_{ori} denote the parameter set of the original actor-critic networks, i.e., $W_{ori} = [\theta^{\pi}, \theta^{Q}]$. Let L_{ori} denote the value set of the original actor-critic networks, i.e., $L_{ori} = [\pi, Q]$. Likewise, the parameter set and value set of the target networks are denoted as $W_{tar} = [\theta^{\pi'}, \theta^{Q'}]$ and $L_{tar} = [\pi', Q']$, respectively.

Let $L_{ori}(W_{ori})$ represent the value function of the original actor-critic networks concerning the weight parameter W_{ori}. W_{ori} are initialized to the same parameters when time slot $n = 0$. Further, when the slot $n > 0$, W_{ori} evolves according to the typically stochastic gradient descent performed by the value function $L_{ori}(W_{ori}(n))$ over its original dataset D_u with user's request, which is estimated by

$$W_{ori}(n+1) = W_{ori}(n) - \gamma \nabla_{D_u} L_{ori}(W_{ori}(n)) \tag{19}$$

where γ is the step-size of gradient descent, and $\nabla_{D_u} L_{ori}(W_{ori}(n))$ is the gradient of value function $L_{ori}(W_{ori}(n))$ over the dataset D_u.

Let σ_u denote the gradient divergence of user u at slot n. The difference is calculated by

$$\sigma_u = \|\nabla_{D_u} L_{ori}(W_{ori}) - \nabla_D L_{tar}(W_{tar})\| \tag{20}$$

where $\nabla_{D_u} L_{ori}(W_{ori})$ is the gradient of the original networks over D_u with certain user's request, and $\nabla_D L_{tar}(W_{tar})$ represents the gradient of the target networks over the edge's dataset D.

The edge server collects all gradients of users at N slots where $n = \kappa N$ and $\kappa = \{1, 2, \ldots\}$. The original actor-critic networks estimate their gradients over the time window $[(\kappa-1)N+1, \ldots, \kappa N]$ for user's request, i.e., $\sum_{\tau=(\kappa-1)N+1}^{\kappa N} \nabla_{D_u} L_{ori}(W_{ori}(\tau))$. Whereas the target actor-critic networks update their parameters using the gradients of the original networks. The parameter updates of the target networks is

$$W_{tar}(\kappa N) = W_{tar}((\kappa-1)N) - \gamma \frac{\sum_{u\in U} |D_u| (\sum_{\tau=(\kappa-1)N+1}^{\kappa N} \nabla_{D_u} L_{ori}(W_{ori}(\tau)))}{|D|} \tag{21}$$

Similar to [34], we adopt the weighted average of all user's gradient divergences to describe the edge's gradient divergence σ at slot n. σ is calculated by

$$\sigma = \frac{\sum_{u\in U} |D_u| \sigma_u}{|D|} \tag{22}$$

Herein, σ is the global gradient divergence for edge server, which can keep the distributed actor-critic learning method convergent.

Combined with Definition 2, we observe that Eq. (17) measures the difference between the task-offloading ratio of a user and the average ratio of the edge over the global dataset. Note that the task-offloading ratio can be increasingly fair for different users with the decrease of q^n. Based on $\mathcal{L}_\infty$ norm, q^n can help to yield the upper bound of the gradient divergence. The fairness can be also derived by the calculation of the edge server. Given the fairness measure q^n, the gradient divergence is strictly upper bounded, as stated in the following lemma.

Lemma 1: We consider the value functions of $L_{ori}(W_{ori})$ and $L_{tar}(W_{tar})$ are ρ-Lipschitz for each user u. Given the fairness measure η^n in Eq. (17), the gradient divergence δ is upper bounded at each time slot n such that

$$\sigma \le B\eta^n \nabla^{max}, \forall n \in \mathbb{N} \tag{23}$$

where ∇^{max} is the maximum gradients based on the $\mathcal{L}_2$ norm during the actor-critic learning.

Proof. Recall that the gradient is a linear operator, the gradient divergence σ_u for user u in Eq. (20) can be rewritten as

$$\begin{aligned}\sigma_u &= \|\sum_{b\in B} \frac{|D_u^b|}{|D_u|} \nabla_{D_u^b} L_{ori}(W_{ori}) - \sum_{b\in B} \frac{|D^b|}{|D|} \nabla_{D^b} L_{tar}(W_{tar})\| \\ &\le \sum_{b\in B} \|\frac{|D_u^b|}{|D_u|} \nabla_{D_u^b} L_{ori}(W_{ori}) - \frac{|D^b|}{|D|} \nabla_{D^b} L_{tar}(W_{tar})\|\end{aligned} \tag{24}$$

where D_u^b stands for the dataset of offloading task b for user u, and D^b denotes the dataset of task b offloaded to the edge, i.e., $D^b = \bigcup_{u\in U} D_u^b$. The inequation above holds because of the subadditive property of the norm.

As assumed, the value functions of $L_{ori}(W_{ori})$ and $L_{tar}(W_{tar})$ are ρ-Lipschitz and are subjected to $\|L_{ori}(W_{ori}) - L_{ori}(W'_{ori})\| \le \rho\|W_{ori} - W'_{ori}\|$ for any W_{ori}, W'_{ori}. The value functions of the original actor-critic networks and the target ones are identical in the aspect of the network structure and model during the training process, i.e., $L_{ori}(W_{ori}(n)) = L_{tar}(W_{tar}(n))$ for $n = \kappa N$. For $n = \kappa N + \tau$ with $\tau = \{1, 2, \ldots, N-1\}$. We have $L_{tar}(W_{tar}(n)) = L_{ori}(W_{ori}(\kappa N)) - \gamma \sum_{t_0=1}^{\tau} \nabla_D L(W(\kappa N + t_0))$. $L_{ori}(W_{ori}(n)) = L_{ori}(W_{ori}(\kappa N)) - \gamma \sum_{t_0=1}^{\tau} \nabla_{D_u} L_{ori}(W_{ori}(\kappa N + t_0))$. Hence, the approximated error is upper bounded by

$$\begin{aligned}&\|L_{ori}(W_{ori}(n)) - L_{tar}(W_{tar}(n))\| \\ &\le \rho\|\gamma \sum_{t_0=1}^{\tau} [\nabla_{D_u} L_{ori}(W_{ori}(\kappa N + t_0)) \\ &- \nabla_D L_{tar}(W_{tar}(\kappa N + t_0))]\| \\ &\le \rho\gamma [\| \sum_{t_0=1}^{\tau} \nabla_{D_u} L_{ori}(W_{ori}(\kappa N + t_0))\| \\ &+ \| \sum_{t_0=1}^{\tau} \nabla_D L_{tar}(W_{tar}(\kappa N + t_0))\|] \\ &\le \rho\tau\nabla^{max} \le \rho N \nabla^{max}\end{aligned} \tag{25}$$

where the first inequation holds due to the ρ-Lipschitz property. The second is due to the subadditive inequation of the norm. The third holds because of $\|\nabla_D L_{tar}(W_{tar})\| \le \nabla^{max}$.

According to Definition 2, the gradient aims to confine the value function $L_{tar}(W_{tar})$ over the data samples D of the edge, denoted as $\nabla_D L_{tar}(W_{tar})$. The average of the gradients over D is denoted by the equation $\nabla_D L_{tar}(W_{tar}) = \frac{1}{D} \sum_{w\in D} \nabla_w L_{tar}(W_{tar})$ where $\nabla_w L_{tar}(W_{tar})$ is the gradient of the value function based on a data sample $w \in D$ [31]. Provided that there are sufficient data samples for learning, the gradient only relies on the distribution of data instead of the number of samples. Thus, the gradient of user's value function over D_u^b, i.e., $\nabla_{D_u^b} L_{ori}(W_{ori}) = \frac{1}{D_u^b} \sum_{w\in D_u^b} \nabla_w L_{ori}(W_{ori})$, is expected to conform to the gradient over D^b, i.e., $\nabla_{D^b} L_{ori}(W_{ori}) = \frac{1}{D^b} \sum_{w\in D^b} \nabla_w L_{ori}(W_{ori})$. Therefore, it is reasonable to assume that the gradient for certain task b of user u is consistent with the gradient over all tasks of user u. That is, $\nabla_{D_u^b} L_{ori}(W_{ori}) \simeq \nabla_{D^b} L_{ori}(W_{ori})$. Since $\nabla_D L_{tar}(W_{tar}) \le \nabla^{max}$, the Eq. (20) can be further estimated by

$$\begin{aligned}\sigma_u &\le \nabla^{max} \sum_{b\in B} |\frac{|D_u^b|}{|D_u|} - \frac{|D^b|}{|D|}| \\ &= \nabla^{max} \sum_{b\in B} |\frac{\overline{a_{u,b}^n}}{\sum_{b\in B} a_{u,b}^n} - \frac{\overline{\sum_{u\in U} a_{u,b}^n}}{\sum_{u\in U, u\in U} a_{u,b}^n}| \\ &= \nabla^{max} \sum_{b\in B} |\frac{\overline{a_{u,b}^n}}{\sum_{b\in B} a_{u,b}^n} - \frac{D_b^{ave}}{\sum_{b\in B} D_b^{ave}}|\end{aligned} \tag{26}$$

where the last equality holds since tasks offloaded to the edge has the same ratio as the average ratio of the dataset in edge [35]. Given Eq. (17), we can yield $\sigma_u \le \nabla^{max} B\eta^n$, and substitute it into Eq. (20) to conclude the proof of Eq. (26). □

From the above theoretical analysis, it is concluded that Algorithm 1 utilizing the enhanced actor-critic reinforcement learning can be converged after the initialization of certain parameters. Next, massive simulations are carried out to further demonstrate its advantages over the benchmark approaches, in terms of task-processing delay, power efficiency, and convergence time.

5. Experience-driven and randomized simulations

5.1. Evaluation setup

We consider a 300 m × 300 m square area with the different number of users. Each user can produce different tasks (services) to process in a time slot with $N = 60$. Similar to [36] and [37], the ARM Cortex-M based IoT devices are employed, which are regarded as the ground users. The energy consumption for the local tasks processed is 141 mW.

In our implementation, a 5-layer fully-connected neural networks with the feed-forward function is used to serve as the actor and critic networks. The neurons of the input layer is the number of dimensions concerning the environment state. The second and third layer both contain 64 neurons. The ReLU (Rectified Linear Unit) [38] is acted

Table 3
Parameter setting in experiments.

Parameters	Values		Parameters	Values
$l(b,n)$	[5, 10]	Cycles	β	0.9
f_b	[0.5, 0.7]	Gbits	ϵ	0.99
c_u	[0.5, 2]	GHz	ξ^{π}	0.001
c_m	10	GHz	ξ^{Q}	0.01
e_{loss}	10^{-9}	J/cycle	λ	0.01

as an activation function before the final output layer which utilizes the softmax for activation. The activation function of softmax is to ensure the sum of output values equals one. The empirical settings obtained by the extensive simulations are outlined in Table 3, where the parameter of f_b set to the range of [0.5 0.7] Gbits intends to inspire the user to offload tasks in order to measure the performance of our proposed algorithms. Some big applications, in general, need to offload their tasks to the edge such as online gaming and virtual reality. The simulation parameters are mostly derived from [39]. The model is trained on the server with 2.3 GHz 4 cores CPU and 32GB random access memory. The power consumption of the smart IoT device is measured by the monitor in the battery.

The OTS with the following other alternatives are compared:

- ACRL: Actor-critic reinforcement learning method is an online approach to tackle the real-time problems of task processing. Different from it, our proposed OST method considers energy efficiency.
- Dynamic programming termed as DP: The main idea of DP is to divide a problem into sub-problems and address all sub-problems to acquire the optimal solutions.
- Heuristic search algorithm termed as HSA: It is one of the benchmark approaches to search the optimally approximated solution by the searching space. The approach of HSA is often an offline method.
- Lagrange duality termed as LD: The method of LD aims to solve the linear programming problem based on the accurate model. Whereas it is tricky to cope with the model-free problem.

The following metrics are employed to evaluate the performance of OST.

- Service response time: It involves the time of processing tasks locally, the delay of the partial task offloaded to MEC and the round-trip transmission latency between users and MEC.
- Users' power efficiency: It is measured by the consumed amount of energy when the user's device deals with a unit of its workload.
- Convergence time: It is a quantified metric for measuring the efficiency of a certain algorithm. It is various in different scenes. It also weighs the feasibility of the algorithm.

The parameter λ in our settings is set to 0.01 for controlling the update rate. The reason is illustrated in Fig. 2. The influence of the task-processing delay on the number of tasks under different update rates are shown in Fig. 2(a). Note that, when the update rate is set to 0.01, the delay of a task is the lowest among the three different values of λ despite the increase in the number of tasks in a time slot. The reason is that the update rate impacts the learning procedure of the actor and critic networks. If it is too small, making the learning process slow, and rising the time duration of the processing tasks. On the contrary, it boosts the learning procedure. While it easily suffers from neglecting the optimal solution. The learning procedure cannot converge until the optimal solution is obtained. Therefore, the delay of task processing at $\lambda = 0.1$ is still higher than that at $\lambda = 0.01$. Similarly, Fig. 2(b) exhibits the impact of reward on the number of tasks to be processed under different update rates in a slot. It is observed that the energy efficiency of $\lambda = 0.01$, overall, performs the best than that of the other values.

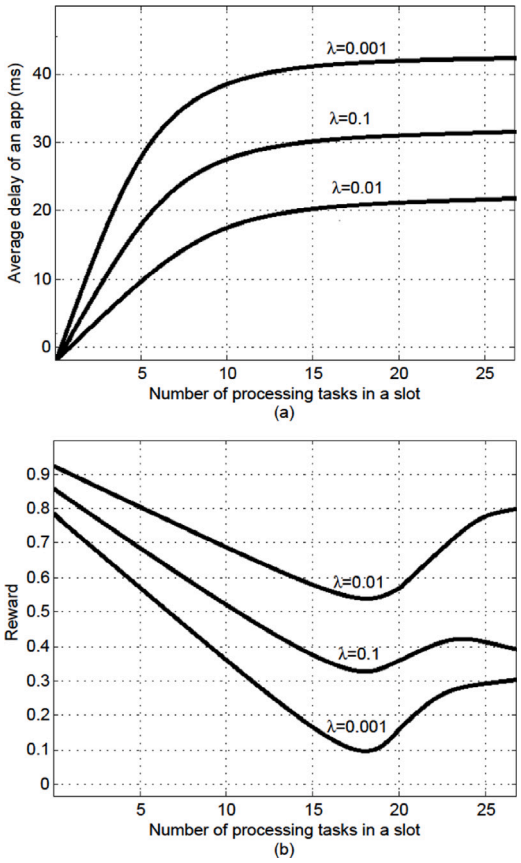

Fig. 2. Training results of OST versus iterations during the training.

5.2. Comparisons with other alternatives

There are 4000 task-processing samples for driving the model-free training so as to derive the solution of the two-dimensional variables. The training results of OST are described in Fig. 3, where Fig. 3(a) shows the average delay of the processing tasks for a user versus epochs, and the loss values of the actor networks and the critic ones versus iterations are exhibited in Fig. 3(b) and Fig. 3(c), respectively. We observe that the loss values between the actor networks and the critic ones are approaching to zero after 1000 epochs. Meanwhile, in Fig. 3(a), it is noted that the average delay of the processing tasks is in the long-term optimal state even if there exist some negligible fluctuations. In other words, before the convergence of the algorithm, the loss values of the actor and critic networks are in the oscillation that means the learning procedure is unstable. Because online learning is a trial-and-error process via the accumulative experience. The average delay of the processing tasks for a user is reduced by 71.42% after experiencing 1000 epochs.

The comparison of the different energy efficiency, the average delay for a user, and the convergence time with the different number of the processing tasks in a slot under different approaches are sketched as Fig. 4. In Fig. 4(a), it is clearly found that the curve of the LD method decreases linearly, suggesting that it is low efficiency to tackle some real-time tasks and a mass of tasks to be processed in a time slot. The curves of DP and HSA are first descending and then slowly go up, showing that the DP and HSA approaches can work well for the small number of tasks. However, they will become worse when the number of processing tasks is larger than 15. In contrast, the curve of the learning-enabled approaches of ACRL and OST are smoothly rising with the increase of the number of the processing tasks in a slot, which implies that the learning-enabled methods perform better than the approaches

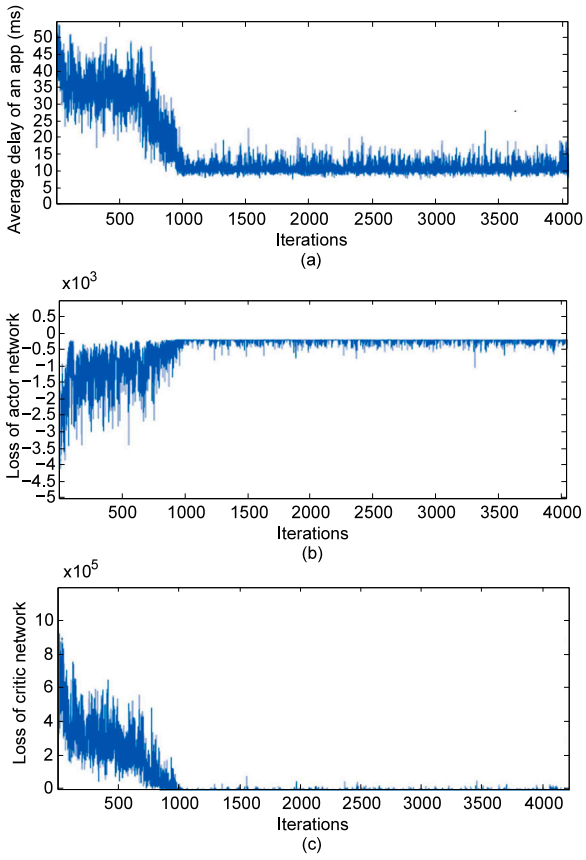

Fig. 3. Training results of OST versus iterations during the training.

without the learning algorithm when the number of tasks is larger than 18. Furthermore, the improvement of the overall energy efficiency is up to 20.5%. Because the proposed OST has taken these factors into account such as the device's energy consumption, the fairness of the assignment of the communication capacity, and the provisioning for the offloading percentage, which enhances the accuracy of reward. The sinuous curve of HSA in Fig. 4(a) indicates that the exhaustive search algorithm is unstable with the growth of the number of the processing tasks in a slot when compared with the curve of DP. Because its search space is extended.

The trends of the two curves of OST and ACRL approaches are first descending and then keeping stable in Fig. 4(b). Compared with them, the trends of the other approaches are increasing with the growth of the number of processing tasks in a slot. This phenomenon shows that the learning-enabled approaches are more applicable to the situations with a larger number of tasks from the long-term view. In terms of the small scale of tasks to be processed in a time slot, especially, within the number of around 20 tasks in a slot, the strategies of OST and ACRL are time-consuming. Because the learning procedure is needed in the initial phase, and the size of training samples, to some extent, has an influence on the leaning time. That is, the small size of samples may cause the unsatisfied training results. The solver of LD to address the linear programming problem has the advantage when the number of tasks is small in a slot. Whereas the average delay of LD rapidly becomes worse than the other approaches in Fig. 4(b). Because it cannot work out some real-time tasks especially for a large number of the processing tasks. The methods of HSA and DP perform better than that of OST and ACRL before the number of processing tasks up to around 20. However, they become worse with the continuous growth of tasks, which means HSA and DP cannot obtain the continuously optimal solvers even if they can cope with the larger number of tasks. On the contrary, OST and ACRL exhibit well under this circumstance. Compared with ACRL, the average delay of OST approaches is reduced by 37.5% for one user.

The convergence time of OST and ACRL is first increasing and then decreasing. Finally, it keeps unchanged with the increase in the number of tasks in Fig. 4(c). Obviously, the approaches of DP, HSA, and LD converge faster than that of OST and ACRL when the number of the processing tasks in a slot is small. It implies that those methods are suitable for the small size of tasks. Overall, the convergence time of OST becomes the lowest among the other approaches after the number of tasks increases to around 17. Further, it keeps 16 ms, which is the lowest in the other approaches. These results are consistent with the results in Fig. 3(a) and Fig. 4(b). Therefore, it is concluded that the approach of the online tailored solver with energy efficiency has significant advantages over the benchmark approaches, in terms of task-processing delay, power efficiency, and convergence time.

5.3. Real world application

We have made experiments with 30 users in a building around 300 m × 300 m square area. Each user generates 10 tasks at a fixed time slot. Other parameters' settings are referred to Table 3. We first fixed the variable $p_{u,b}$, setting it to be 200 MC/s (MC=10^6 cycles) [36] and [37] in some practical scenes (i.e., channel communication capacity is fixed in some commercial application), and then we make it a block box to flexibly adjust its tailored requirements. Herein, since the comparison with other alternatives in Section 5.2 has been discussed, we set about discussing the approaches of OST and ACRL carefully, both of which utilize the actor-critic reinforcement learning.

The discussions on the comparison of the energy efficiency and the average delay with iterations under the premise of the respectively known or unknown variable of $p_{u,b}$ are conducted. As shown in Fig. 5 from (a) to (d), it is observed that, compared with the fixed $p_{u,b}$ in Fig. 5(a) and (c), the energy efficiency of OST and ACRL with the unfixed $p_{u,b}$ in Fig. 5(b) is respectively improved by about 25% and 16.7% when they are in convergence. Correspondingly, the average delay of OST and ACRL with the unfixed $p_{u,b}$ in Fig. 5(d) is respectively reduced by about 56.3% and 40% when they are in convergence. It

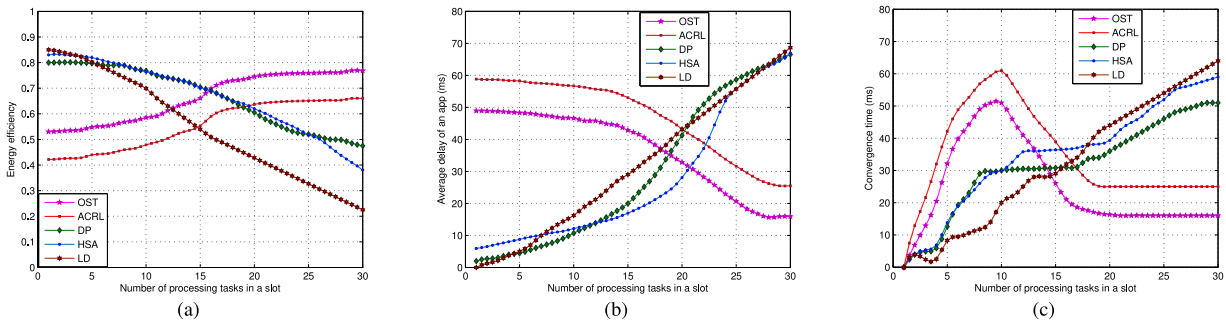

Fig. 4. (a) Energy efficiency, (b) average delay for a user, and (c) convergence time versus the number of processing tasks in a slot.

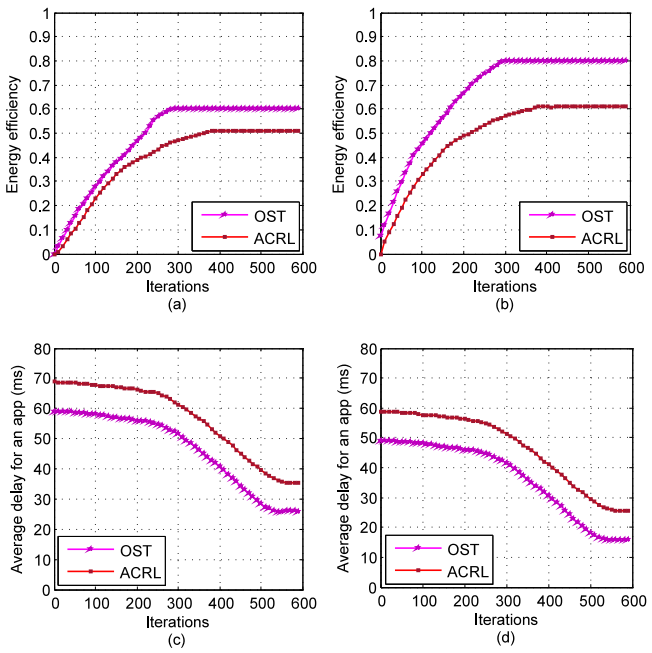

Fig. 5. Comparison of convergence time and system performance with different approaches.

suggests that the allocation of the unfixed communication capacity performs better than that of the fixed value of $p_{u,b}$ if the approaches use the learning-based methods to complete the assignment. Note that the convergence time is embodied by the iterations of OST and ACRL. The curves of them stay stable at the iterations of around 300 and 400, respectively. Overall, the convergence time of OST is faster than that of ACRL due to the consideration of the tailored energy efficiency for users.

6. Conclusion

The proliferation of mobile IoT devices and data-intensive computing is driving up a significant demand for services with low latency and high energy efficiency. Different from existing works focusing on some canonical models incorporating the linear programming model, game-theoretical model, and so forth, we first propose a novel unifying task-processing orchestrator that is an online tailored solver for learning the model-free model in consideration of the partial and binary task offloading. Further, the energy efficiency of IoT devices is elaborately designed in a distributed manner according to the real-world settings, shifting the research of the conventional architecture to the user-oriented IoT devices. Besides, the first attempt is made on an innovative actor-critic reinforcement learning approach taken the energy efficiency into account, which derives an asymptotically optimal solution. Extensive simulation analysis demonstrates our proposed approach outperforms the state-of-the-art in terms of the task-processed delay, energy efficiency, and convergence time.

In future work, since the deep learning models are driven by big data, the black-box learning approach leveraging some specifically isolated and small data will be further explored to circumvent oscillation.

Declaration of competing interest

No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.comnet.2021.107875.

Acknowledgment

All authors have read and contributed to the manuscript.

References

[1] L. Yu, Z. Li, Y. Zhong, Z. Ji, J. Liu, When QoE meets learning: A distributed traffic-processing framework for elastic resource provisioning in HetNets, Comput. Netw. 167 (2020) 106904.
[2] Z. Ning, P. Dong, X. Kong, F. Xia, A cooperative partial computation offloading scheme for mobile edge computing enabled internet of things, IEEE Internet Things J. 6 (3) (2019) 4804–4814.
[3] L. Pu, X. Chen, G. Mao, Q. Xie, J. Xu, Chimera: An energy-efficient and deadline-aware hybrid edge computing framework for vehicular crowdsensing applications, IEEE Internet Things J. 6 (1) (2019) 84–99.
[4] G.S. Park, W. Kim, S.H. Jeong, H. Song, Smart base station-assisted partial-flow device-to-device offloading system for video streaming services, IEEE Trans. Mob. Comput. 16 (9) (2017) 2639–2655.
[5] S. Bi, Y.J. Zhang, Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading, IEEE Trans. Wireless Commun. 17 (6) (2018) 4177–4190.
[6] M. Chen, Y. Hao, Task offloading for mobile edge computing in software defined ultra-dense network, IEEE J. Sel. Areas Commun. 36 (3) (2018) 587–597.
[7] F. Zhou, Y. Wu, R.Q. Hu, Y. Qian, Computation rate maximization in UAV-enabled wireless-powered mobile-edge computing systems, IEEE J. Sel. Areas Commun. 36 (9) (2018) 1927–1941.
[8] T. Bai, J. Wang, Y. Ren, L. Hanzo, Energy-efficient computation offloading for secure UAV-edge-computing systems, IEEE Trans. Veh. Technol. 68 (6) (2019) 6074–6087.
[9] Z. Yan, M. Peng, M. Daneshmand, Cost-aware resource allocation for optimization of energy efficiency in fog radio access networks, IEEE J. Sel. Areas Commun. 36 (11) (2018) 2581–2590.
[10] Y. Geng, Y. Yang, G. Cao, Energy-efficient computation offloading for multicore-based mobile devices, in: Proc. of IEEE INFOCOM, 2018, pp. 46–54.
[11] Q. Qi, J. Wang, Z. Ma, H. Sun, Y. Cao, L. Zhang, J. Liao, Knowledge-driven service offloading decision for vehicular edge computing: A deep reinforcement learning approach, IEEE Trans. Veh. Technol. 68 (5) (2019) 4192–4203.
[12] N. Cheng, F. Lyu, W. Quan, C. Zhou, H. He, W. Shi, X. Shen, Space/aerial-assisted computing offloading for IoT applications: A learning-based approach, IEEE J. Sel. Areas Commun. 37 (5) (2019) 1117–1129.
[13] Z. Yan, M. Peng, C. Wang, Economical energy efficiency: an advanced performance metric for 5G systems, IEEE Wirel. Commun. 24 (1) (2017) 32–37.
[14] Y. Wu, L.P. Qian, K. Ni, C. Zhang, X. Shen, Delay-minimization nonorthogonal multiple access enabled multi-user mobile edge computation offloading, IEEE J. Sel. Top. Sign. Proces. 13 (3) (2019) 392–407.
[15] J. Xu, L. Chen, P. Zhou, Joint service caching and task offloading for mobile edge computing in dense networks, in: Proc. of IEEE INFOCOM, 2018, pp. 207–215.
[16] Y. Zhou, Z.M. Fadlullah, B. Mao, N. Kato, A deep-learning-based radio resource assignment technique for 5g ultra dense networks, IEEE Netw. 32 (6) (2018) 28–34.
[17] C. Liu, Z. Chen, J. Tang, J. Xu, C. Piao, Energy-efficient UAV control for effective and fair communication coverage: A deep reinforcement learning approach, IEEE J. Sel. Areas Commun. 36 (9) (2018) 2059–2070.
[18] Y. Xiao, M. Krunz, Distributed optimization for energy-efficient fog computing in the tactile internet, IEEE J. Sel. Areas Commun. 36 (11) (2018) 2390–2400.
[19] I. Grondman, L. Busoniu, G.A.D. Lopes, R. Babuska, A survey of actor-critic reinforcement learning: Standard and natural policy gradients, IEEE Trans. Syst. Man Cybern. 42 (6) (2012) 1291–1307.
[20] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous Control with Deep Reinforcement Learning, in: Proc. of ICLR, 2016.
[21] K.G. Vamvoudakis, F.L. Lewis, Online actor–critic algorithm to solve the continuous-time infinite horizon optimal control problem, Automatica 46 (5) (2010) 878–888.
[22] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C.H. Liu, D. Yang, Experience-driven Networking: A Deep Reinforcement Learning based Approach, in: Proc. of IEEE INFOCOM, 2018, pp. 1871–1879.
[23] G. Rishwaraj, S.G. Ponnambalam, C.K. Loo, Heuristics-based trust estimation in multiagent systems using temporal difference learning, IEEE Trans. Cybern. 47 (8, SI) (2017) 1925–1935.
[24] P. Mach, Z. Becvar, Mobile edge computing: A survey on architecture and computation offloading, IEEE Commun. Surv. Tutor. 19 (3) (2017) 1628–1656.

[25] L. Pu, X. Chen, J. Xu, X. Fu, D2D Fogging: An energy-efficient and incentive-aware task offloading framework via network-assisted D2D collaboration, IEEE J. Sel. Areas Commun. 34 (12) (2016) 3887–3901.
[26] T.Z. Oo, N.H. Tran, W. Saad, D. Niyato, Z. Han, C.S. Hong, Offloading in hetnet: A coordination of interference mitigation, user association, and resource allocation, IEEE Trans. Mob. Comput. 16 (8) (2016) 2276–2291.
[27] C. Chen, B. Wang, R. Zhang, Interference hypergraph-based resource allocation (IHG-RA) for NOMA-integrated V2X networks, IEEE Internet Things J. 6 (1) (2019) 161–170.
[28] J. Zhu, Y. Song, D. Jiang, H. Song, A new deep-Q-learning-based transmission scheduling mechanism for the cognitive Internet of Things, IEEE Internet Things J. 5 (4) (2018) 2375–2385.
[29] H.V. Hasselt, A. Guez, D. Silver, Deep Reinforcement Learning with Double Q-Learning Artificial intelligence, in: Proc. of AAAI, 2016, pp. 2094–2101.
[30] L. Yu, Z. Li, J. Liu, R. Zhou, Resources Sharing in 5G Networks: Learning-Enabled Incentives and Coalitional Games, 2020, https://doi.org/10.1109/JSYST.2019.2958890.
[31] L. Bottou, Large-scale machine learning with stochastic gradient descent, in: Proc. of COMPSTAT, Springer, 2010, pp. 177–186.
[32] L. Yu, Z. Li, J. Liu, R. Zhou, Resources sharing in 5G networks: Learning-enabled incentives and coalitional games, IEEE Syst. J. (2019) https://doi.org/10.1109/JSYST.2019.2958890.
[33] S. Ruder, An overview of gradient descent optimization algorithms, 2016, arXiv preprint arXiv:1609.04747.
[34] S. Wang, T. Tuor, T. Salonidis, K.K. Leung, C. Makaya, T. He, K. Chan, When edge meets learning: Adaptive control for resource-constrained distributed machine learning, in: Proc. of IEEE INFOCOM, 2018, 63–71.
[35] Q. Mao, F. Hu, Q. Hao, Deep learning for intelligent wireless networks: A comprehensive survey, IEEE Commun. Surv. Tutor. 20 (4) (2018) 2595–2621.
[36] ARM, ARM Cortex-M for Beginners. [Online]. Available:, 2017, https://community.arm.com/cfs-file/__key/telligent-evolutioncomponents-attachments/01-2142-00-00-00-00-52-96/White-Paper-_2D00_-Cortex_2D00_M-for-Beginners-_2D00_-2016-_2800_finalv3_2900_.pdf.
[37] A. Hussain, Energy consumption of wireless IoT nodes, M.S. Thesis, Dept. Inf. Secur. Commun. Technol., Norwegian Univ. Sci. Technol., Trondheim, Norway, NTNU, 2017.
[38] Y.B. I. Goodfellow, A. Courville, Deep Learning, MIT Press, 2016, http://www.deeplearningbook.org/.
[39] X. Ma, J. Liu, H. Jiang, Resource allocation for heterogeneous applications with device-to-device communication underlaying cellular networks, IEEE J. Sel. Areas Commun. 34 (1) (2016) 15–26.

Li Yu received her M.S. degree from Zhengzhou University, China, in 2016, and since 2019, she has been funded by China Scholarship Council as a visiting Ph.D. student at Simon Fraser University, Canada. She is currently a Ph.D. candidate in the School of Computer Science, Wuhan University, China. Her current research interests include machine learning, wireless networks, Internet of Things, Mobile Edge Computing, mobile network optimization, and Software Defined Network.

Zongpeng Li (SM'12) received his B.E. in Computer Science from Tsinghua University in 1999, and his Ph.D. from the University of Toronto in 2005. He has been affiliated with the University of Calgary and then Wuhan University. His research interests are in computer networks, network algorithms, and cloud computing. He received the Outstanding Young Computer Scientist Prize from the Canadian Association of Computer Science, and a few Best Paper Awards from conferences in related fields.

Jiangchuan Liu (F'17) received the B.Eng. degree (cum laude) in computer science from Tsinghua University, Beijing, China, in 1999, and the Ph.D. degree in computer science from The Hong Kong University of Science and Technology, in 2003. He is currently a University Professor with the School of Computing Science, Simon Fraser University, BC, Canada. He is the Steering Committee Chair of the IEEE/ACM IWQoS, from 2015 to 2017, and the TPC Co-Chair of the IEEE IC2E 2017 and the IEEE/ACM IWQoS 2014. He serves as an Area Chair for the IEEE INFOCOM, ACM Multimedia, and the IEEE ICME. He has served on the Editorial Boards of the IEEE Transactions on Big Data, the IEEE Transactions on Multimedia, the IEEE Communications Surveys and Tutorials, the IEEE Access, the IEEE Internet of Things Journal, Computer Communications, and Wireless Communications and Mobile Computing (Wiley). (Based on document published on 9 December 2018).

Ruiting Zhou has been an Associate Professor in the School of Cyber Science and Engineering at Wuhan University since June 2018. She received a M.S. degree in telecommunications from Hong Kong University of Science and Technology, Hong Kong, in 2008, a M.S. degree in computer science from University of Calgary, Canada, in 2012 and her Ph.D. degree in 2018 from the Department of Computer Science, University of Calgary, Canada. Her research interests include cloud computing, machine learning and mobile network optimization. She has published research papers in top-tier computer science conferences and journals, including IEEE INFOCOM, IEEE/ACM TON, IEEE JSAC, IEEE TMC. She also serves as a reviewer for journals and international conferences such us IEEE JSAC, IEEE TMC, IEEE TCC, IEEE TWC, IEEE Transactions on Smart Grid and IEEE/ACM IWQOS. She held NSERC Canada Graduate Scholarship, Alberta Innovates Technology Futures (AITF) Doctoral Scholarship, and Queen Elizabeth ll Graduate Scholarship from 2015 to 2018.