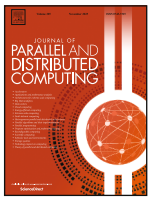
Journal of Parallel and Distributed Computing 209 (2026) 105191

Contents lists available at ScienceDirect

Journal of Parallel and Distributed Computing

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



Reload: Deep reinforcement learning-based workload distribution for collaborative edges

Yu Liang [a,*], Jidong Ge [b,c], Jie Wu [d], Sheng Zhang [b,e], Shiwu Wen [b,c], Bin Luo [b,c]

[a] School of Computer and Electronic Information/School of Artificial Intelligence, Nanjing Normal University, China
[b] State Key Laboratory for Novel Software Technology, Nanjing University, China
[c] Software Institute, Nanjing University, China
[d] Center for Networked Computing, Temple University, Philadelphia, USA
[e] School of Computer Science, Nanjing University, China

ARTICLE INFO

Keywords:
Edge computing
Collaborative edges
Deep reinforcement learning

ABSTRACT

Edge computing is one of the promising technologies that aim to enable timely computation at the network edge. Major service providers started to deploy geographically-distributed edge servers several years ago. A major challenge in geographically distributed edges is task scheduling, i.e., how to assign various tasks submitted by mobile users to distributed edges so as to optimize some metric. However, it is not easy to perform task scheduling in geographically-distributed edges. We observed that there are three major challenges in designing an efficient task scheduling solution in dynamic edge environments: heterogeneous tasks, dynamic edge networks, and heterogeneous edge servers. In this paper, we pursue a black-box solution for task scheduling in the collaborative edge environment while not relying on detailed analytical performance modeling. We propose Reload, an intelligent deep reinforcement learning-based task scheduler. Reload learns a policy purely based on the known information, without foreseeing the future. Reload depicts its policy as a neural network that maps "raw" observations to scheduling actions. During training, Reload starts out knowing nothing and gradually learns to make better scheduling decisions through reinforcement, in the form of reward signals for past decisions. Reload leverages Advantage Actor Critic to train the policy network. We evaluate Reload using extensive simulations.

1. Introduction

There is an explosive growth in the volume of the data generated at the edge of the Internet in recent years. For example, a high-definition, traffic monitoring camera can generate several terabytes of data in a day [1]. Besides, more and more computation-intensive applications such as interactive gaming, real-time video analytics and virtual or augmented reality are becoming more and more popular at the network edge; what is worse, these applications usually demand low delay. Cloud computing seems to be unable to timely handle such large amounts of data, since, on the one hand, transmitting such large amounts of data to centralized compute clusters incurs a non-negligible latency. On the other hand, linearly-increased computation capabilities in centralized compute clusters cannot match the data generated in the network edge with an exponential growth.

Edge computing is one of the promising technologies that aim to enable timely computation at the network edge [2]. Major service providers started to deploy geographically-distributed edge servers several years ago. For example, over 1400 edge servers are deployed by Google [3]. With these geo-distributed edge servers, end users can have low-latency edge service anytime and anywhere, which greatly mitigates the effect of large amounts of data on backbone networks and centralized data centers.

Task scheduling is key to geographically distributed edges, i.e., how to assign various tasks submitted by mobile users to distributed edges so as to optimize some metric, e.g., the average task response time, the number of tasks finished before their deadlines. Previous works fall short in performing task scheduling in geographically-distributed edges. Most of them [1,4–6] focused on offloading bursty task requests to one single server or device. Some other studies on edge collaboration aimed to highlight its advantages in the device-to-device environments. For example, the work in [7] proposed a method to achieve energy-efficient collaborative task execution. These studies probably fail since most of them hardly consider the dynamics of the edge environments.

* Corresponding author.
E-mail addresses: liangyu@njnu.edu.cn (Y. Liang), gjd@nju.edu.cn (J. Ge), jiewu@temple.edu (J. Wu), sheng@nju.edu.cn (S. Zhang), mf20320174@smail.nju.edu.cn (S. Wen), luobin@nju.edu.cn (B. Luo).

https://doi.org/10.1016/j.jpdc.2025.105191
Received 9 November 2022; Received in revised form 31 October 2025; Accepted 31 October 2025
Available online 5 November 2025

It is not easy to perform task scheduling in geographically-distributed edges. We observed that there are three major challenges in designing an efficient task scheduling solution in dynamic edge environments. Firstly, user tasks are heterogeneous and dynamic in terms of task workload[1], task deadline, task input size, etc. The heterogeneity of user tasks prohibit some efficient design of algorithms. Secondly, the edge networks are often dynamic in terms of bandwidth, especially the upload bandwidth from the end users to edge servers (which are usually deployed together with a cellular base station or an access point). Thirdly, the workloads of different servers may differ significantly, since edge servers are geographically distributed and such geographical difference makes edge servers preferred by different end users.

In this paper, we consider the following general scenario. A number of heterogeneous edge servers are deployed in a local area. Each edge server is associated with an access point (AP). An AP could be a cellular base station, a WiFi router, or any other hardware that can receive and send messages. Hence, these edge servers are connected wirelessly. These APs and edge servers form the *collaborative edge environment*, since they collaboratively serve end users by executing the tasks offloaded from end users and sending analytics results back to them too. In such a scenario, mobile users can move anywhere and anytime. Each user can offload its task to the collaborative edge environment for quick execution. Usually, a user submits its task via the nearest AP from itself. Note that, as we mentioned before, the bandwidth between each user and the AP is often dynamic over time. In this paper, we aim to minimize the average response time of all tasks offloaded to the collaborative edge environment. The response time of a task is defined as the time difference between when the task is submitted to the collaborative edge environment and when the task analytics results are sent back to the user. When a user submits its task to the edge environment, the edge environment must determine to which edge servers this task should be offloaded and by what ratio, i.e., how much workload of this task should be offloaded to each edge server. To this end, we must intelligently perform the task scheduling.

However, task scheduling in collaborative edge environments optimally is hard for the following reasons. Firstly, since the completion time of a task relies on not only itself but also other tasks in the pending queues on each server, thus it requires information of future tasks to solve the current optimization problem. Secondly, since an edge server schedules tasks with the first come first served (FCFS) strategy, the current decision also affects the completion time of future tasks. Thirdly, the completion time of a task depends on the workload of pending queues, thus it is impossible to give a closed-form expression for a completion time.

Therefore, we pursue a black-box solution for task scheduling in the collaborative edge environment while not relying on detailed analytical performance modeling. We choose deep reinforcement learning (DRL) [10,11] as it has shown its advantages in solving many hard problems in various research fields. However, integrating DRL into the proposed problem is challenging due to limited scheduling budget and large decision space. Firstly, for a scheduling problem, there exists a tradeoff between scheduling quality and scheduling overhead. High-quality scheduling can effectively reduce task completion time, but at the cost of consuming a long scheduling time. Secondly, in our problem, we have to not only decide the target edge servers but also the percentages of offloaded workloads, leading to a large decision space. These two intrinsically intertwined challenges together complicate the considered problem.

We propose Reload, an intelligent deep reinforcement learning-based task scheduler that is customized for task offloading in collaborative edge environments. Reload learns a policy purely based on the known information, without foreseeing the future. Reload depicts its policy as a neural network that maps "raw" observations to scheduling actions. The neural network incorporates a rich diversity of observations into the scheduling policy in a scalable manner. During training, Reload starts out knowing nothing and gradually learns to make better scheduling decisions through reinforcement, in the form of reward signals for past decisions. Reload leverages Advantage Actor Critic (A2C) [12] to train the policy network, which takes the edge network situation, server statistics, and task features as inputs and selects an optimal action through the output (e.g., action distribution). We evaluate Reload using extensive simulations.

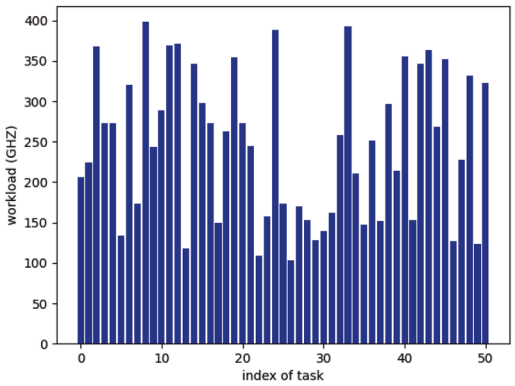


Fig. 1. Heterogeneous workloads.

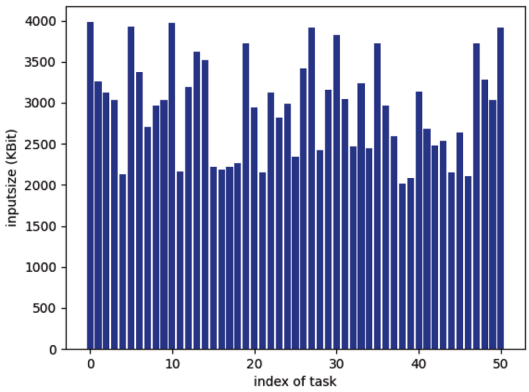


Fig. 2. Heterogeneous input sizes.

The main contributions are summarized as follows.

- We consider a scenario in which edge servers are wireless connected and collaboratively execute tasks offloaded from various end users.
- We design and implement Reload. Although the tasks, edge networks, and edge servers have their own heterogeneities and dynamics, they have hidden laws which should help us perform scheduling if we can learn the laws.
- We conduct extensive simulations to evaluate the performance of Reload. The evaluation results confirm the effectiveness of Reload.

The rest of the paper is organized as follows. We motivate our study in Section 2. We introduce the task scheduling problem in Section 3. We then present Reload in Section 4. Evaluation is given in Section 5. We survey related work in Section 6. We discuss limitations and future work in Section 7 and conclude the paper in Section 8.

2. Motivation

In this section, we motivate our study by showing the dynamics of the edge environments from three different and possibly orthogonal aspects, namely, heterogeneous tasks, dynamic edge networks, and heterogeneous edges.

[1] Task workload is quantified by the amount of computations (i.e., the total number of CPU cycles) required to accomplish a task [8,9].

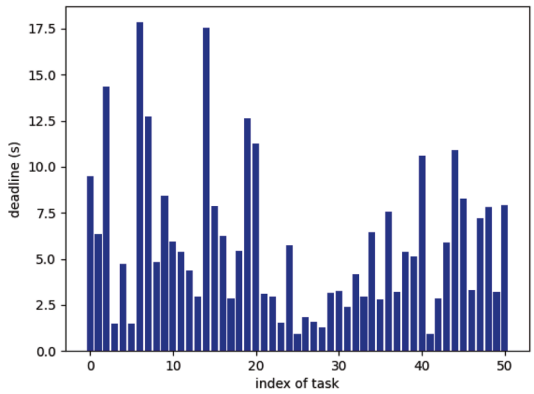


Fig. 3. Heterogeneous deadlines.

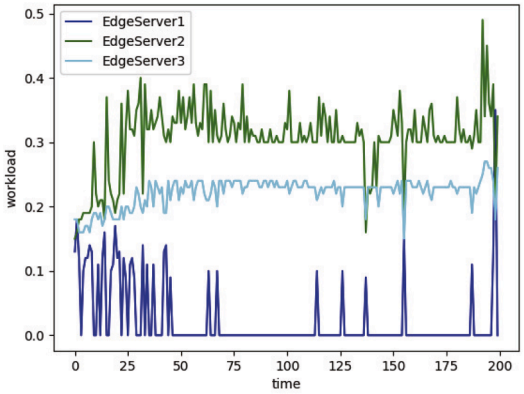


Fig. 4. The pending workloads in edge servers are often imbalanced.

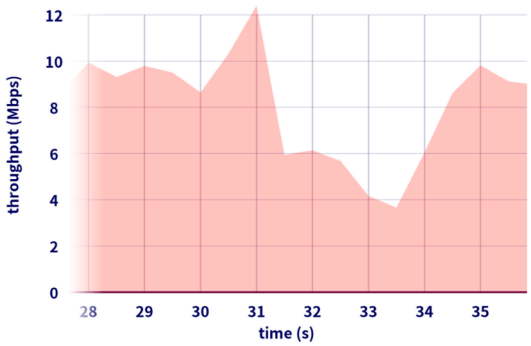


Fig. 5. The upload link bandwidth in the Mahimahi trace [13].

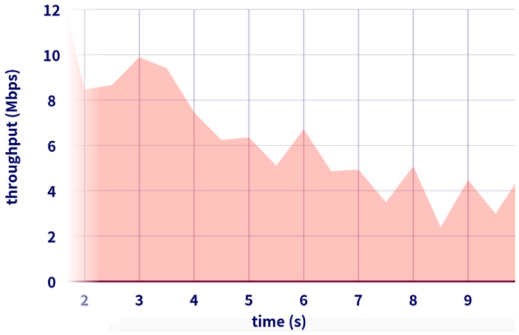


Fig. 6. The download link bandwidth in the Mahimahi trace [13].

2.1. Heterogeneous tasks

Edge environments are often open and dynamic compared with the cloud environment. Mobile users in an edge environment can submit any types of tasks that have heterogeneous task workloads, input sizes, and deadlines. To demonstrate this, we collected the task information submitted to an IoT agent in the power IoT system of State Grid Jiangsu. In this system, various end devices connect to one or several IoT agents and all the IoT agents connect to the IoT management platform located at the remote cloud. An IoT agent can be seen as an edge server and it can help end devices to run computation-intensive tasks. The task statistics is shown in Figs. 1–3.

Fig. 1 shows the heterogeneous workloads of various tasks collected in a typical edge environment. We see that the largest task in terms of workload could be nearly 4× larger than the smallest one, and about 30 % percent of tasks have over 130 % of the average workload of all the 50 tasks. Similar observations can be found in Figs. 2 and 3 For example, in Fig. 3, the longest deadline is nearly 12× larger than the smallest one.

Such heterogeneity of the user tasks brings challenges to efficient task scheduling in the collaborative edge cloud environment. If all tasks have similar workloads, then we can pre-find some optimal or near-optimal scheduling strategy to achieve a good scheduling performance, so as to save the scheduling overhead for each single user task. However, these heterogeneous tasks make this approach not appropriate.

2.2. Dynamic edge networks

In our paper, we assume mobile users freely move within the target area in the collaborative edge cloud environment. There are two sources of the time-varying upload bandwidths from each mobile user to the edge servers: one is the user mobility, and the other is the time-varying nature of the upload and download links between devices and base stations.

Figs. 5 and 6 show the time-varying upload and download bandwidths of two ATT-LTE network traces in the Mahimahi project [13]. We can see from these two figures that, there is hardly any pattern that can be learned to model the time-varying bandwidths. For example, time periods with either extremely low or high bandwidths seem to randomly emerge along the time dimension; the upload link bandwidth has little relationship with the download link bandwidth.

These unpredictable bandwidths further increase the difficulty of devising an efficient scheduling approach in such a dynamic and open environment.

2.3. Heterogeneous edges and imbalanced workloads

Edge servers are also heterogeneous in terms of their processing speeds, storage sizes, and so on. For example, Nvidia AGX Xavier has 512-core Volta GPU with 64 Tensor cores and 32 GB 256-bit LPDDR4x, while Raspberry Pi4 has only 1.5 Ghz Cortex-A72 with 4 GB LPDDR4 SDRAM at most.

Besides, the workloads in different edge servers in a typical scenario are also heterogeneous. Fig. 4 shows the imbalanced workloads of three edge servers (i.e., IoT agents) in the power IoT system of State Grid Jiangsu [14]. We see that, even these three edge servers have tremendously different workloads over time, and the workload of each single edge server also fluctuates over time. Such imbalance also makes the task scheduling in a collaborative edge cloud environment not trivial.

The above illustrations are far from exhaustive. However, we believe these examples illustrate a few representative scenarios. Thus, studying the task scheduling problem in a collaborative edge cloud environment should probably help us develop techniques that hold across many similar scenarios.

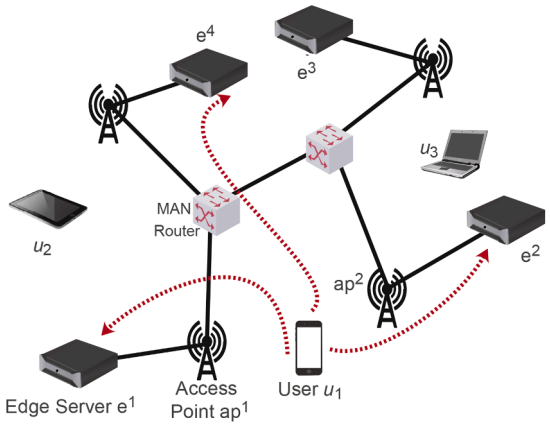


Fig. 7. An example of the scenario, in which edge servers are connected by MAN routers and users freely move and submit their tasks to the collaborative edge cloud.

3. System model and problem formulation

In this section, we formulate the problem of task scheduling in the collaborative edge environments. We also provide a formal mathematical formulation of the problem.

We divide time into discrete time slots, each of which has a duration that matches the timescale at which task scheduling decisions can be generated. We envision the following scenario. In a local area network, there are in total N edge servers, e_1, e_2, ..., and e_N, deployed in this scenario. These edge servers are usually deployed with APs, i.e., each edge server is connected to an AP. When a user submits its task to the collaborative edge environment, the controller decides how to schedule the task so as to minimize the average response time of all submitted tasks. Here, we assume there is a centralized controller which could be any edge server, since all the edges constitute a collaborative edge cloud. The edges in a collaborative edge cloud could exchange and update their status periodically, and thus, keep the centralized controller knowing the entire system state. When the controller produces the scheduling results via Reload, it sends them to related edge servers. Then, these edge servers can perform admission control or dispatch user tasks. The delay between two edge servers e^i and e^j is $g^{i,j}$. Take Fig. 7 for example, there are $N = 4$ edge servers connected by several MAN routers and access points.

We assume the computing capacity of edge server e^i is c^i. Following a previous work [15], we assume that each edge server can process only one task at any time. It is easy to see that there is no benefit in processor-sharing with respect to the sum queueing-and-execution delay of the tasks. The reason is as follows. In our work (i.e., Reload) and many previous studies, the delay of a task contains the queueing delay and the execution delay, in which the former denotes the delay of a task waiting in the queue and the latter denotes the delay of execution the task workload. Therefore, as long as the processor (e.g., CPU, GPU) is fully utilized, there is no benefit in processor-sharing with respect to the sum queueing-and-execution delay of the tasks. Furthermore, if a single task cannot utilize the full computational capacity of an edge server every time, we can simply take an edge server as two logical edge servers.

An edge server may receive more than one tasks to execute. In this case, an edge server uses first come first served (FCFS) to decide the order of tasks to execute. More specifically, for each edge server, it maintains a FCFS queue which stores the tasks that are assigned to it. The sum of the workloads in the pending queue of edge server e^i at time t is denoted by $p^i[t]$.

Modern edge servers usually have plenty of storage, e.g., Nvidia Jetson AGX Xavier has up to 64GB eMMC 5.1 storage [16], while the input data of a user task is often less than several gigabytes. Thus, we do not consider the storage constraints in this work.

Mobile end users move in the given scenario. Denote by u_i the i-th user. We assume that the time between two consecutive tasks from the same user u_i follows the exponential distribution with mean λ_i, since the exponential distribution is often used to simulate a wide variety of real life events. Without loss of generality, let us safely suppose the current task of mobile end user u_i is task f_i. The total workload size of task f_i is w_i. Its deadline is d_i, which means the task f_i should be finished before its deadline d_i, otherwise the task analytics results would be useless. We assume the release time of task f_i is r_i. The input data size of task f_i is z_i. Its completion time is t_i. Note that, the completion time is unknown before the task is finished; we use this notation for simplicity.

In fact, the mobile end device of each user u_i has a local computing capacity h_i, which is relatively small compared with the computing capacity of an edge server. When a user plans to offload its task to the collaborative edge cloud, the user can reserve partial workload of the task for its local processing.

In this paper, we assume that the workload of a user task is fine-grained and permits arbitrary partitioning [17–19], and that the output of a user task, i.e., the analytics results, can be constructed by consolidating partial outputs from pieces of the workload. In fact, many tasks in reality have these two properties, e.g., large matrix-vector products and large file compression. These tasks have the characteristic that they can be arbitrarily partitioned into any number of load fractions. These load fractions may or may not have precedence relations. For example, in the case of Kalman filtering applications, the data is arbitrarily divisible but there may exist precedence relation among these data segments or load fractions. On the other hand, if the load fractions do not have precedence relations, then each load fraction can be independently processed. These latter type of loads are the ones which are of interest to us. Applications which satisfy this divisibility property include processing of massive experimental data, image processing applications like feature extraction and edge detection, and signal processing applications like extraction of signals buried in noise from multidimensional data collected over large spans of time, computation of Hough transforms, and matrix computations.

With these two reasonable assumptions, when the collaborative edge cloud performs task scheduling, the total workload of a user task can be freely distributed among multiple edge servers in any ratio, and the final output of the user task can be derived by gathering partial outputs from each piece of workload. Besides, following many previous works [19–21], we assume the output of a user task is relatively small compared with the input data size, therefore, we do not take the time for a user to collect the output of its task into consideration. However, it is not hard to extend our solution to handle this.

As we mentioned before, each end user can freely move, thus its physical location may change over time. At any time, a user can be covered by more than one access point, e.g., when a user is located at the intersection of the coverage areas of two neighboring assess points. We denote by $b_i^j[t]$ the upload bandwidth between user u_i and edge server e^j at time t. These bandwidths are time-varying.

Main notations are summarized in Table 1 for quick reference.

When there is a task, the global/centralized controller must decide to which edge and by what ratio to offload it, e.g., offload 20 % of the workload of task f_7 to e_3. Therefore, we use x_i^j to denote the percentage of the workload of task f_i that is offloaded to e^j. Without causing any confusion, we use x_i^0 to denote the workload processed locally by the local computing capacity of user u_i. Note that

$$\sum_{j=0}^{N} x_i^j = 1. \tag{1}$$

For example, there are four edge servers in Fig. 7 Suppose user u_1 generates a task f_1. A feasible assignment could be $x_1^0 = 0.2$, $x_1^1 = 0.3$, $x_1^2 = 0.4$, $x_1^3 = 0$, and $x_1^4 = 0.1$, which means 20 % percent of the

Table 1
Main notations for quick reference.

Symbol	Meaning	Symbol	Meaning
N	number of edge servers	c^i	computing capacity of i-th edge server
e^i	i-th edge server	$g^{i,j}$	delay between edge servers e^i and e^j
u_i	i-th user	h_i	local computing capacity of user u_i
f_i	task of user u_i	λ_i	average time difference between the arrivals of two consecutive tasks
r_i	release time of f_i	$b_i^j[t]$	upload bandwidth between user u_i and server e^j at time t
w_i	workload of f_i	E_i	set of servers that should process some workload from task f_i
d_i	deadline of f_i	t_i^j	time duration that server e^j finishes task f_i
z_i	input data size of f_i	$p^j[t]$	workloads in the pending queue of server e^i at time t
t_i	completion time of f_i	x_i^j	percentage of workload of f_i offloaded to e^j

workload of task f_1 would be processed locally, while 30 % of the workload would be offloaded to edge server e_1, and so on.

When a user offloads its task to the collaborative edge cloud, the edge servers are not necessarily directly connected to the user. In this case, the workload is offloaded to an edge server that is not directly connected to the user along the shortest path (can be computed using some shortest path algorithm, e.g., Dijkstra's algorithm)) in terms of delay. The red dotted lines in Fig. 7 illustrate the offloading paths. Note that, Dijkstra's algorithm is used to compute the shortest path between a user and an edge server, not to decide which edge server to offload.

Let us put it more formally. Denote by E_i the set of edge servers that should process some workload from task f_i, i.e.,

$$E_i = \{e^j \,|\, x_i^j > 0, 1 \le j \le N\}. \tag{2}$$

For each $e^j \in E_i$, we can simply use some shortest path algorithm (e.g., Dijkstra's algorithm) to find the shortest path from user u_i to edge server e^j. We use $\delta_i^j[t]$ to represent the delay of transmitting the input data along the shortest path from u_i to e^j, that is, $\delta_i^j[t] = \frac{z_i}{b_i^j[t]}$. Note that, each edge server has a queue that stores the pending workloads. As we have mentioned before, each edge server processes the workloads in the pending queue in a FCFS manner. Therefore, for edge server $e^j \in E_i$, when it receives the workload of task f_i, the time is $r_i + \delta_i^j[t]$, which is the sum of the release time and the delay. Thus, edge server e^j can finish the workload of task f_i by $\frac{p^j[r_i+\delta_i^j[t]]+x_i^j w_i}{c^j}$. We denote by t_i^j the time that edge server e^j finishes the workload of task f_i, that is

$$t_i^j = \delta_i^j[t] + \frac{p^j[r_i + \delta_i^j[t]] + x_i^j w_i}{c^j}. \tag{3}$$

Without causing any confusion, we use t_i^0 to represent the time that user u_i itself finishes its local workload, that is

$$t_i^0 = \frac{x_i^0 w_i}{h_i}. \tag{4}$$

Therefore, the completion time t_i of f_i is

$$t_i = \max_{j=0, e^j \in E_i} t_i^j. \tag{5}$$

We aim to minimize the average response time of all user tasks before the deadline, which is equivalent to maximizing the difference between d_i and t_i. We use T to denote the length of the time of interest. Therefore, the problem investigated in this paper can be formulated as follows:

$$\max \quad \sum_{0 \le r_i \le T} (d_i - t_i) \qquad \text{[TSiCE]} \tag{6a}$$

$$\text{s.t.} \quad t_i = \max_{j=0, e^j \in E_i} t_i^j, \qquad \forall i \tag{6b}$$

$$t_i^0 = \frac{x_i^0 w_i}{h_i}, \qquad \forall i \tag{6c}$$

$$t_i^j = \delta_i^j[t] + \frac{p^j[r_i + \delta_i^j[t]] + x_i^j w_i}{c^j}, \qquad \forall i \tag{6d}$$

$$\sum_{j=0}^{N} x_i^j = 1, \qquad \forall i \tag{6e}$$

$$\text{var.} \quad x_i^j \in [0, 1], \qquad \forall i, j \tag{6f}$$

This problem is called Task Scheduling in Collaborative Edges, or TSiCE for short in this paper. Note that TSiCE is not a linear programming problem. TSiCE requires information of future tasks to solve. Remember that, the completion time of a task depends on the workload of pending queues, while the pending queues may change due to the arrivals of future tasks. Specifically, $p^j[r_i + \delta_i^j[t]]$ in Eq. (6) denotes the amount of workloads in the pending queue of server e^i at time $(r_i + \delta_i^j[t])$, in which r_i is the release time and $\delta_i^j[t]$ represents the delay of transmitting the input data along the shortest path from u_i to e^j, thus, it is impossible to give a closed-form expression for a completion time.

Solving this problem optimally is not easy for the following reasons. Firstly, since the completion time of a task relies on not only itself but also other tasks in the pending queues on each server, thus it requires information of future tasks to solve the current optimization problem. Secondly, since an edge server schedules tasks with FCFS, the current decision also affects the completion time of future tasks. Thirdly, the completion time of a task depends on the workload of pending queues, thus it is impossible to give a closed-form expression for a completion time. These intrinsically intertwined challenges together complicate the TSiCE problem. Therefore, we would like to use deep reinforcement learning (DRL) to solve this problem.

4. Deep reinforcement learning-based workload distribution

In this section, we first introduce the basic learning mechanism in Reload. Then, we show how to use deep reinforcement learning (DRL) to solve the TSiCE problem and present the details about the state space, action space, and reward signal.

4.1. Basic learning mechanism

Deep reinforcement learning (DRL) learns an effective policy from the past experiences based on the current state and instant reward. In Reload, a RL-agent is responsible for interacting with the environment and performing task scheduling decisions, where RL stands for reinforcement learning. In Reload, the concept of environment is in fact an abstraction of the edge servers, mobile end users, and edge networks. The RL-agent cannot observe the entire environment status, that is, it only knows part of all the information of the environment. At each time slot t, the RL-agent observes a state that represents the status of the environment and chooses an action based on the specific policy. After this action is performed (i.e., the task scheduling decision is performed), the state would change, and the change should represent the effect of the action on the environment. In this paper, the RL-agent is the collaborative edge cloud controller.

4.2. Details of reload

Due to the lack of future knowledge and the state transition probability matrix, as well as the discrete decision space, we propose the model-free DRL-based Reload, which is trained using a state-of-the-art actor-critic DRL model called A2C. We introduce the details as follows.

4.2.1. State space

The state is the observation of the RL-agent (i.e., the collaborative edge cloud controller) from the environment. The RL-agent aims to constantly learn policies from historical information to approach the perspective of the God (i.e., have future and global knowledge). Thus a comprehensive state is critical to the decision-making efficiency. We take the information of the task itself, the dynamic networks, and the edge environment into consideration.

- Task workload w_i, task deadline d_i, task release time r_i, and task input data size z_i. The TSiCE problem deals with online task scheduling in the collaborative edge cloud environments, thus, it is necessary to take the characteristics of each task into consideration. These heterogeneous tasks may prefer different offloading servers or different offloading ratios. These heterogeneous release times may let tasks encounter different statuses of the environments, e.g., different edge network conditions and different pending workloads in edge servers.
- The upload bandwidths from user u_i to all edge servers. Note that, each user can freely move, and its location changes over time, which finally makes the upload bandwidth from each user to all edge servers change. Remember that, we use $b_i^j[t]$ to represent the upload bandwidth from user u_i to edge server e^j at time t. Also note that, each user may not be covered by all edge servers; when it is out of the coverage of any edge server, we set the upload bandwidth from the user to the edge server to be 0. This bandwidths information reflects the dynamic networks, which should be considered in the decision making process.
- The sizes of the pending workloads in each edge server. Remember that, each edge server maintains a FCFS queue that stores the pending workloads that have been assigned to it but have not been finished. The size of the pending queue of an edge server reflects how popular an edge server is. The TSiCE problem investigated in this paper is an online problem; it is hard since we cannot acquire future information about the network, mobility, and task statistics. The size of the pending queue of each edge server can help us capture such dynamics to some extent. Note that, p_0^t represents the sum of the workloads in the pending queue of mobile end user u_i.

Note that, the above states do not directly reflect the mobility of users. But the user mobility is captured in $b_i^1[t], \cdots, b_i^N[t]$, since $b_i^j[t]$ is 0 if u_i is not covered by e^j.

4.2.2. Action space

In our proposed Reload, a RL-agent needs to take an action for task scheduling when receiving a new user task or a new state. As we mentioned before, we use x_i^j to denote the percentage of the workload of task f_i that is offloaded to e^j. Without causing any confusion, we use x_i^0 to denote the workload processed locally by the local computing capacity of user u_i. Therefore, the action space is represented by $x_i^0, x_i^1, \cdots, x_i^N$.

We know that, on one hand, the larger the action space is, the more time the deep reinforcement learning-based method needs. Therefore, the action space should not be too large; otherwise, the large action space may make the training of Reload a little bit longer and the inference result of Reload not so accurate.

On the other hand, if we only allow a very small action space, Reload may not be able to explore the solution space efficiently and probably cannot arrive at the relatively good task scheduling decision.

In our current design, each x_i^j can only choose a value from the set $\{0, 0.1, 0.2, \cdots, 1.0\}$, i.e., each x_i^j can be only one of the eleven values. Later, in the extensive performance evaluations we will see, Reload with this choice about the action space performs better than several state-of-the-art baselines.

Note that, the current choice about the action space can be easily modified to other ranges or sets.

4.2.3. Reward

Once applying an action to a state, the RL-agent would receive an instant reward from the environment. Recall that in the problem formulation, our optimization objective is to maximize the total difference between the deadlines and the completion times, i.e., $(d_i - t_i)$, that is, we would like to make each task be finished as quickly as possible. Therefore, when $(d_i - t_i)$ is large, the reward $reward_i$ should be large.

However, as we mentioned before, tasks have heterogeneous characteristics, e.g., different tasks may have different workloads and deadlines. Directly maximizing the difference between the deadlines and the completion times may let Reload focus on optimizing any single task. Therefore, we define the reward as follows:

$$reward_i = \frac{d_i - t_i}{d_i - r_i}, \tag{7}$$

in which the denominator $(d_i - r_i)$ is the difference between the deadline and the release time.

4.3. DRL Model training methodology

We pursue a black-box solution for task scheduling in the collaborative edge environment while not relying on detailed analytical performance modeling.

We propose Reload, an intelligent deep reinforcement learning-based task scheduler that is customized for task offloading in collaborative edge environments. Reload learns a policy purely based on the known information, without foreseeing the future. Reload depicts its policy as a neural network that maps "raw" observations to scheduling actions. The neural network incorporates a rich diversity of observations into the scheduling policy in a scalable manner. During training, Reload starts out knowing nothing and gradually learns to make better scheduling decisions through reinforcement, in the form of reward signals for past decisions. Reload leverages A2C to train the policy network, which takes the edge network situation, server statistics, and task features as inputs and selects an optimal action through the output (e.g., action distribution).

In this subsection, we introduce the details about our training methodology of Reload.

Firstly, we optimize the action space to avoid the curse of dimensionality. Reload adopts A2C to output an action a_t for a given state s_t. In Reload, the action space is represented by $x_i^0, x_i^1, \cdots, x_i^N$; even though we only allow each x_i^j to be one of the values in the set $\{0, 0.1, 0.2, \cdots, 1.0\}$, the number of possible actions could be approximately 11^N, which is extremely large. To solve this problem in the training process of Reload, we fix the dimensionality of an action a_t to be $N + 1$, and we take the policy π as the input of the function $softmax$[2]. Therefore we can directly output the percentage of workloads of a given task offloaded to each edge server. That is, given a state s_t and corresponding action a_t, the output is

$$y = softmax(\pi(a_t|s_t)). \tag{8}$$

Secondly, we use the policy gradient method to train Reload. The policy gradient $J(\theta)$ can be calculated as follows:

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[\sum_{t \in T} \nabla_\theta \log(\pi_\theta(s_t, a_t)) A(s_t, a_t)], \tag{9}$$

in which $A(s_t, a_t)$ is called the advantage function that represents the gap between the expected reward when we deterministically select a_t at state s_t following π_θ, and the expected reward for actions drawn from policy π_θ. Its specific form is

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t), \tag{10}$$

[2] It can convert a tuple of K real numbers into a probability distribution of K possible outcomes. It is a generalization of the logistic function to multiple dimensions, and is used in multinomial logistic regression [22].

in which $Q(s_t, a_t)$ is the state action value function and $V(s_t)$ is the state value function. In practice, we do not need to estimate these two functions simultaneously; we can use only $V(s_t)$ to achieve the same effect. More specifically, we can estimate $V(s_t)$ based on the temporal difference method, and we can get the simplified gradient formula as follows:

$$\nabla_\theta J(\theta) = E_{\pi_\theta}\left[\sum_{t\in T} \nabla_\theta \log(\pi_\theta(s_t, a_t))(r + \gamma V(s_t') - V(s_t))\right], \tag{11}$$

where $r + \gamma V(s_t') - V(s_t)$ is the unbiased estimation of $A(s_t, a_t)$, which is also called *td_error* and generated by the critic network. Based on the above formulas, the parameter of the actor network is updated as follows:

$$\theta \leftarrow \theta + \alpha \sum_{t\in T} \nabla_\theta \log(\pi_\theta(s_t, a_t))(r + \gamma V(s_t') - V(s_t)), \tag{12}$$

in which α is the learning rate and γ is the reward discount.

To make the results returned by the actor network more accurate, we use the critic network to estimate td_error. In the critic network, the parameter is updated as follows:

$$\delta \leftarrow \delta - \alpha \sum_{t\in T} \nabla_\delta (r + \gamma V(s_t') - V(s_t))^2. \tag{13}$$

We introduce a hyper-parameter clip_bound for the following two reasons. Firstly, in our work, the completion time of a task includes two parts, including the transmitting delay from the task source to an edge server and the processing delay on an edge server. When the percentage of workloads assigned to an edge server is less than clip_bound, the transmitting delay may dominate the overall completion time; in such case, redistributing this small percentage of workload to another edge server or the task source itself may be better. We also evaluate the effect of using the branch and bound strategy in our extensive evaluations.

Secondly, in traditional A2C algorithm, the output is a probability distribution function, and we can get the specific action by sampling. However, Reload modifies the form of the output of A2C, hence Reload cannot generate a probability distribution function as the output, which may not be beneficial to exploration and exploitation. To tackle this, we incorporate a trick used in DDPG [23] into Reload. DDPG introduces a Gaussian noise to the output for better exploration. To achieve the same purpose, we introduce a hyper-parameter clip_bound to introduce randomness to the output of Reload. In Reload, when the percentage of workloads assigned to an edge server is less than clip_bound, Reload proactively sets the percentage to zero and redistributes this percentage of workload to the other edge servers. In this sense, the hyper-parameter clip_bound serves the threshold of workload assignment.

Note that, Reload changes the probability distribution function of the action space from the Gaussian distribution in A2C to the uniform distribution. Besides, as we mentioned earlier, we introduced a hyper-parameter clip_bound to Reload for better exploration. Therefore, due to these modifications (or improvements), the policy used in Reload is neither the epsilon-greedy nor sampling by probability in A2C.

The time complexity of DRL involves multiple components, and a precise analysis requires consideration of the network structure, update methods, and parallel architecture. Specifically, the time complexity of DRL involves Actor and Critic Network Structure, Advantage Function Calculation, Policy Gradient Update, Value Function Update, and Parallel Architecture. For example, the time Complexity of Forward Pass can be analysed as follows: For a neural network with L layers, each containing n_i neurons (where i ranges from 1 to L), the time complexity of a forward pass is approximately $O(\sum_{i=0}^{L} n_i n_{i-1})$. In practice, the shared feature extraction layers significantly reduce the overall computational cost.

In summary, the overall time complexity per update step is $O(2 \cdot \sum_{i=0}^{L} n_i n_{i-1})$, accounting for both the Actor and Critic networks. In practice, the shared feature extraction layers reduce this complexity, and the parallel architecture improves training efficiency without altering the per-update time complexity.

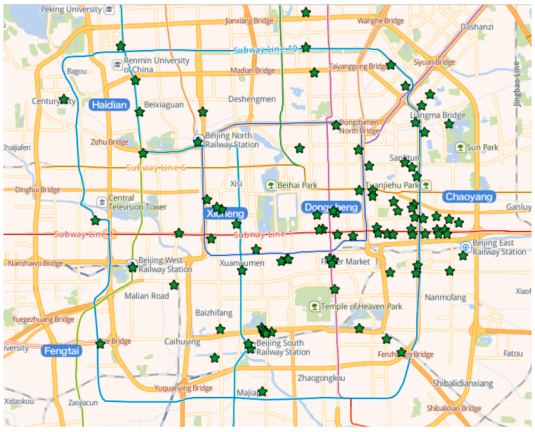


Fig. 8. Locations of 92 Starbucks in Beijing, P.R. China.

Table 2
DRL parameters used in the evaluations.

Symbol	Meaning	Default Value
LR_A	learning rate for actor network	0.001
LR_C	learning rate for critic network	0.001
GAMMA	reward discount	0.900
BATCH_SIZE	batch size	32
MAX_EPISODES	maximum value of episodes	260
MAX_EP_STEPS	maximum value of steps	100
MEMORY_CAPACITY	memory capacity of DDPG	1000
clip_bound	the exploration parameter	0.090

5. Performance evaluation

In this section, we evaluate the performance of Reload using extensive simulations. We first present the evaluation settings. Then we introduce several state-of-the-art baselines for comparison. Finally we show the comparison results in a variety of settings.

5.1. Evaluation settings

We consider a metropolitan area that contains edge servers and users. Similar to a previous study [24–26], we used Starbucks' locations as the locations of edge servers, because the distribution of them in a city usually achieves a decent coverage of users, making them very suitable for placing edge servers. Therefore, we collected the locations of Starbucks within the 4th ring road of Beijing, China, as shown in Fig. 8 We calculated the minimum bounding rectangle of the Starbucks with two sides parallel to a meridian. Then, we extended this rectangle by adding 5km to each side, so as to form the area of interest. Note that, the number of edge servers was fixed, i.e., 92 in the Beijing trace. In our simulations, we may not use all of these locations to place edge servers; instead, we can randomly select a part of them to deploy edge servers. Mobile users can freely move inside the area. Similar to [24–26], the delay between two edge servers is proportional to the Euclidean distance between them in all settings. In our simulations, we assume 1km incurs 1ms. Therefore, when the locations of clients are known, we can calculate the delay between every pair of servers and clients.

In our simulations, the system time starts from 0. We divide time into discrete time slots, each of which has a duration that matches the timescale at which task scheduling decisions can be generated. At the end of each time slot, Reload collects the information and makes the scheduling decision. The learning rate and reward discount are fixed at 0.001 and 0.9, respectively, in our implementation. Main DRL-related parameters are summarized in Table 2.

The upload bandwidth from each mobile end user to the edge servers (more exactly, access points) is time-varying and also depends on the

location of a mobile user. In our simulations, we generate these bandwidths according to the Mahimahi trace [13].

Similar to previous works [8,9,15], the computing capacities of edge servers follow a uniform distribution between 100 and 200. The local computing capacity of each mobile device is uniformly generated from a range, e.g., $[10, 40]$. Note that, the size of the pending workloads in the FCFS queues of edge servers are determined by the specific environments and the scheduling. Each mobile user generates its tasks over time. For mobile user u_i, we assume the time difference between two consecutive tasks follows an exponential distribution with the mean λ_i uniformed distributed in a range $[20, 50]$. The workload of a user task follows a uniform distribution, e.g., $[100, 400]$. The input size of each user task is set according to the Mahimahi trace [13], in which the size is uniformly distributed between 500 kbit and 2000 kbit. For the deadline of a user task, it should not be too large nor too small. If the deadline is too large, then all tasks can be finished before the deadlines, making the scheduling meaningless; otherwise, all tasks cannot be finished before the deadlines, making that even the best scheduling algorithm cannot finish any task before its deadline. In our paper, we generate the deadline d_i of user task f_i uniformly between min and max, where $min = \frac{w_i}{h_i + \sum_{u_i \text{ is covered by } e^j} c^j}$ represents the minimal possible value of deadline and $max = \frac{w_i}{h_i}$ represents the maximal possible value of deadline.

We compare Reload with the following baselines.

- SS-B (Single Server with maximal Bandwidth). Each task is processed only in one edge server. We offload the task to the edge server with the maximal upload bandwidth.
- SS-W (Single Server with minimal Workloads). Each task is processed only in one edge server. We offload the task to the edge server with the minimal pending workloads.
- DS-BW (Double Server with maximal Bandwidth and minimal Workloads). Each task is processed in two edge servers. We offload the task to one server with the maximal bandwidth and to another server with the minimal pending workload.
- MS (Multiple Server). After Reload gives a solution (e.g., offloads 30 % workload to edge server e_1, 35 % to e_7, 15 % to e_9, and 20 % locally), MS uses these ratios (i.e., 30 %, 35 %, 15 %, 20 %) to offload the same task. MS chooses the 3 servers with minimal delays.

5.2. Results

We are interested in evaluating how Reload performs in different settings compared with the other baseline algorithms. In this subsection, we present our evaluation results about the performance of Reload, which is the mean of 10 runs.

5.2.1. Comparison results in terms of reward

In Fig. 9(a), we compare Reload with the other four algorithms in terms of the reward metrics over time. In this figure, the number of edge servers is 10, and the average time difference between two consecutive tasks is uniformly distributed in $[20, 50]$. In general, we see that the reward achieved by Reload is the highest among all algorithms, in particular, Reload achieves a much higher reward than SS-W, SS-B, and DS-BW. The main reason behind the best performance of Reload is that, Reload utilizes the advantage of deep reinforcement learning and thus can efficiently explore the solution space. In contrast, the other baseline algorithms do not take all possible solutions into consideration; for example, in the SS-W algorithm, each user task can only be offloaded to just one edge server, which greatly impacts the performance of SS-W. In addition, we observed that Reload converges after 87 episodes on average.

We also observe that the performance of MS is better than the other three baseline algorithms, and MS achieves almost the same performance as Reload in some cases. The main reason is that MS utilizes partial results from Reload. After Reload computes a task scheduling result, MS knows how many edge servers should be chosen to offload a given task. In fact, Reload greatly helps MS reduce the solution space and makes MS focus on selecting a given number of edge servers. Therefore, MS can obtain a better performance than the other baseline algorithms.

In Fig. 9(b), we change the number of edge servers to 20 compared with Fig. 9(a). We note a few observations. First, the rewards achieved in Fig. 9(b) by each algorithm is better than those in Fig. 9(a). For example, the average rewards achieved by the SS-B are about 0.23 and 0.01, respectively, in Figs. 9(b) and 9(a). The main reason is that, the number of edge servers in Fig. 9(a) is smaller than that in Fig. 9(b); more edge servers means more computing results, which can finally translate into reduced response times. Second, the DS-BW algorithm performs better than SS-W but worse than SS-B. This is because the SS-W algorithm focuses on the size of the pending workload of each edge server, which may be influenced by many factors, such as the arrival of new tasks and the computing capacity of each edge server. In contrast, the SS-B algorithm focuses on the delay, which is directly related to the response time. Therefore, SS-B is hopefully better than SS-W.

In Fig. 9(c), the average time difference between two consecutive tasks is changed to be uniformly distributed in $[40, 70]$ compared with Fig. 9(a). We see similar results as the previous two figures: Reload has the best performance among all five algorithms and SS-W has the worst performance.

5.2.2. Comparison results in terms of the percentage of tasks finished before deadlines

The reward metric represents the total utility we gain from scheduling a user task. We are also interested in a more direct metric, i.e., the percentage of tasks finished before the deadlines. Figs. 10(a), 10(b), and 10(c) show the comparison result in terms of this metric.

In Fig. 10(a), we see that the percentage of tasks finished before the deadlines is as high as 91.15 %, while the other four baseline algorithms can achieve 55.0 % at most. From this figure, we can see that, Reload can not only optimize the scheduling utility in terms of the reward metric, but also maximize the number of tasks finished before the deadlines.

Fig. 10(b) shows the comparison results when the average time difference between two consecutive task arrivals is changed to be uniformly distributed in $[40, 70]$. We see that, when there are less tasks generated in unit time, most of the algorithms perform better than before, e.g., the percentage of tasks finished before the deadlines is 55.0 % and 87.33 % in Fig. 10(a) and 10(b), respectively. The reason behind this phenomenon is not subtle; when there are less tasks and the edge servers remain the same, the probability of a single task being finished before its deadline gets higher. Fig. 10(c) shows similar results and trends.

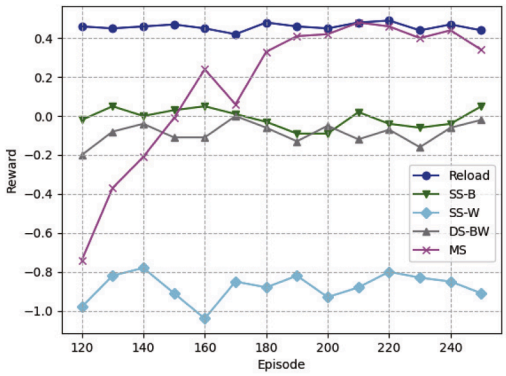
5.2.3. Improvement due to branch and bound

To better explore the solution space, we augmented Reload with the branch and bound strategy to efficiently reduce the solution space, help Reload jump out of the local optimum, so as to obtain a better scheduling output. Fig. 11 shows the effect of the branch and bound. In this figure, 'Reload_nobound' denotes Reload without branch and bound, while 'Reload' denotes the improved version. We can see that, Reload achieves a better reward than Reload_nobound over time.

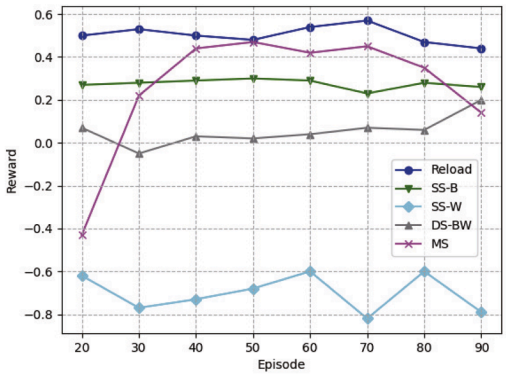
6. Related work

One of the major research fields in edge computing is offloading, which can be seen as the general killer application to the edge computing paradigm. By offloading computation-intensive and possible energy-hungry tasks to nearby edge servers, mobile devices can accelerate the task and save its energy consumption.

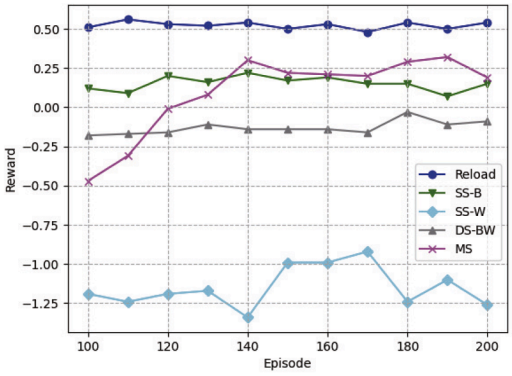
Several studies [27–29] have been proposed to optimize energy consumption on mobile devices by offloading all or partial workloads of a task. For example, Cuervo et al. [30] proposed MAUI that enables automated method-level code offloading; Kosta et al. [27] improved method-level migration by parallelizing method execution using multiple VM



(a) $N = 10$ and λ_i is uniformly distributed between 20 and 50



(b) $N = 20$ and λ_i is uniformly distributed between 20 and 50



(c) $N = 10$ and λ_i is uniformly distributed between 40 and 70

Fig. 9. Reward vs. Episode. The number of edge servers is N. The average time difference between the arrivals of two consecutive tasks is λ_i.

images; Gordon et al. [28] designed thread-level migration techniques based on distributed shared memory.

There are some other studies [5,31–34] that focused on minimizing the completion time of the tasks from mobile end users. Wang et al. [31] considered minimizing the average task latency by intelligently performing edge resource provisioning; Xu et al. [5] took both service caching and offloading into consideration; Wang et al. [24] proposed an efficient solution which is based on the classical uncapacitated facility location problem [35]; Gao et al. [36] studied the service entity placement problem while taking both edge workload and access network congestion into account; Chen and Wang [37] considered computation offloading in a multi-user MIMO system with time-varying channel states.

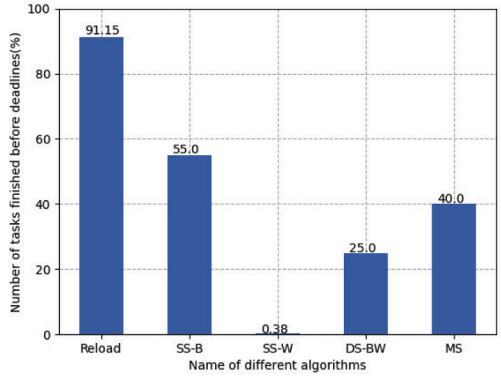
Some recent studies have already considered the edge collaboration [5,38–40]. For example, OREO [5] jointly optimized service caching policies and task offloading strategies. Xu et al. [39] optimized service caching cost using game theory. A hierarchical model with intra-fog and inter-fog resource management was proposed in [41]. VACS [42] maximized video analytics accuracy while reducing the overall energy consumption when offloading. CERP [43] jointly optimized device probing and task offloading and proposed a cost-aware edge resource probing framework based on multi-stage optimal stopping problem. DDPS [44] optimized edge resource utilization based on Stackelberg game [45] when the offloading scenario contains energy harvesting devices. Scrava [46] considered offloading video analytics tasks to edges to support collaborative inference. However, most of them [1,4–6] focused on offloading bursty task requests to one single server or device. Some other studies on edge collaboration aimed to highlight its advantages in the device-to-device environments. For example, the work in [7] proposed a method to achieve energy-efficient collaborative task execution. These studies probably fail since most of them hardly consider the dynamics of the edge environments.

DRL has been widely used in many research fields, e.g., task scheduling for data centers [47,48], traffic engineering [49], multipath TCP routing [50], and task offloading in edge environments [40,51]. However, none of the existing studies observed the dynamics of the edge environments and took the heterogeneity into consideration, which motivates us to propose an intelligent DRL-based solution that performs offloading decisions without foreseeing the future.

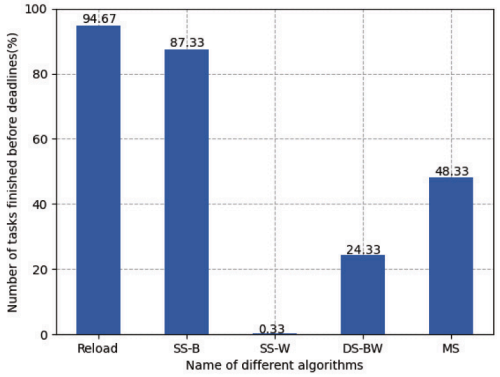
7. Discussions

In this section, we discuss several limitations of Reload and possible future research work directions.

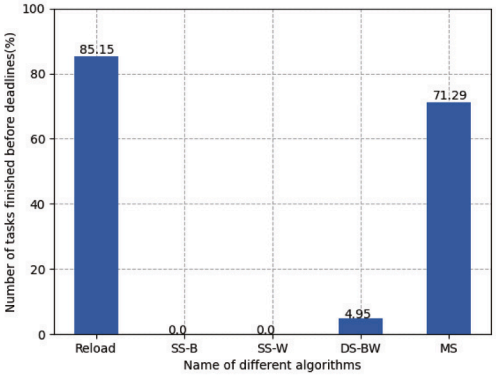
Dynamic edge server set. Reload solves the online task scheduling problem in the collaborative edge cloud environment. In its current design, the set of edge servers in the collaborative edge cloud is fixed. In reality, we may allow an edge device to leave or join the collaborative edge cloud at any time. For example, an edge server crashes due to heavy workload or high temperature, then some engineer repairs it and makes



(a) $N = 10$ and λ_i is uniformly distributed between 20 and 50



(b) $N = 10$ and λ_i is uniformly distributed between 40 and 70



(c) $N = 20$ and λ_i is uniformly distributed between 40 and 70

Fig. 10. The percentage of tasks finished before deadlines. The number of edge servers is N. The average time difference between the arrivals of two consecutive tasks is λ_i.

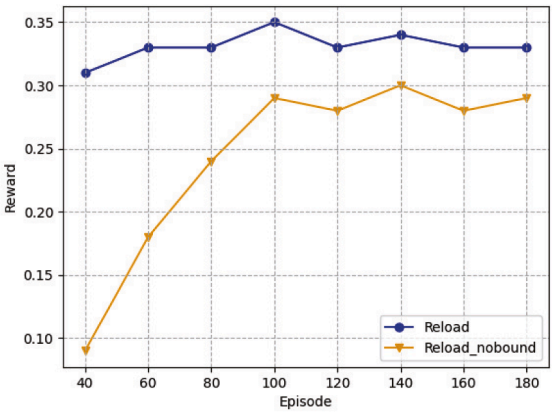


Fig. 11. Reload with branch and bound vs Reload.

it run again. In this case, this edge server leaves the edge cloud first and then joins the edge cloud as a new edge device. However, extending Reload to handle such a dynamic set of edge servers is not easy. The main reason is that, when the set of edge servers changes, the underlying neural models in Reload probably need to change so as to maintain a good performance. Here is one simple strategy that may work. We can pre-train several neural models that correspond to different representative scales of the edge cloud for Reload. When the collaborative edge cloud changes, Reload can choose the most appropriate model to use.

Delay-tolerate task. In its current design, Reload schedules a task right after it arrives. Although this can minimize the completion time of this task, it may prolong the completion time of some future tasks. For example, at time t_1, task f_a arrives at the collaborative edge cloud. We run Reload and get the schedule results, e.g., offloading task f_a to edge servers e_4 and e_5, which are two edge servers that are far away from task f_a, since the pending queue in the edge servers that are near task f_a is almost full. At time $t_1 + 1$, a new task f_b arrives at the collaborative edge cloud, and task f_b is very close to edge servers e_4 and e_5. Besides, the workload of f_b is much smaller than that of f_a. Therefore, if we can let the workload of f_b execute before f_a, then the average response time would be reduced. This example motivates us to take the delay-tolerate tasks into consideration when improving Reload. For each task, when it arrives, Reload can wait for a fixed time before scheduling it. This may bring more opportunities for improving the overall system performance.

More general utilities. The goal of Reload is to maximize the total differences between the deadlines and the completion times of all tasks. In reality, more objectives should be supported in Reload. For example, some tasks may only care about whether the deadline is missed, some tasks may want to minimize the offloading cost. In the future, we plan to investigate supporting more general objectives in Reload.

8. Conclusions

In this paper, we consider the task scheduling problem in the collaborative edge cloud environments. We observed that there are three major challenges in designing an efficient task scheduling solution in dynamic edge environments: heterogeneous tasks, dynamic edge networks, and heterogeneous edge servers. We pursue a black-box solution, Reload, for task scheduling in the collaborative edge environment while not relying on detailed analytical performance modeling. Reload learns a policy purely based on the known information, without foreseeing the future. Reload depicts its policy as a neural network that maps "raw" observations to scheduling actions. During training, Reload starts out knowing nothing and gradually learns to make better scheduling decisions through reinforcement, in the form of reward signals for past decisions. Reload leverages A2C to train the policy network. We evaluate Reload using extensive simulations. The results show that Reload outperforms several state-of-the-art baselines.

CRediT authorship contribution statement

Yu Liang: Writing – review & editing, Writing – original draft; **Jidong Ge:** Supervision; **Jie Wu:** Resources; **Sheng Zhang:** Formal analysis, Conceptualization; **Shiwu Wen:** Validation; **Bin Luo:** Project administration.

Data availability

Data will be made available on request.

Declaration of competing interest

On behalf of all authors, the corresponding author states that there is no conflict of interest in our manuscript titled "Reload: Deep Reinforcement Learning-based Workload Distribution for Collaborative Edges", submitted to Journal of Parallel and Distributed Computing.

Acknowledgments

This work was supported in part by NSFC (62202233), Double Innovation Plan of Jiangsu Province (JSSCBS20220409), and Grant from State Key Laboratory for Novel Software Technology, Nanjing University (KFKT2024B18).

References

[1] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, M.J. Freedman, Live video analytics at scale with approximation and delay-Tolerance, in: Proc. USENIX NSDI 2017, pp. 377–392.
[2] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, Edge computing: vision and challenges, IEEE Internet Things J. 3 (5) (2016) 637–646.
[3] M. Calder, X. Fan, Z. Hu, E. Katz-Bassett, J. Heidemann, R. Govindan, Mapping the expansion of Google's serving infrastructure, in: Proc. SIGCOMM IMC 2013, pp. 313–326.
[4] M.-H. Chen, B. Liang, M. Dong, Joint offloading and resource allocation for computation and communication in mobile cloud with computing access point, in: Proc. IEEE INFOCOM 2017, IEEE, 2017, pp. 1–9.
[5] J. Xu, L. Chen, P. Zhou, Joint service caching and task offloading for mobile edge computing in dense networks, in: Proc. IEEE INFOCOM 2018, IEEE, 2018, pp. 207–215.
[6] Y. Geng, Y. Yang, G. Cao, Energy-efficient computation offloading for multicore-based mobile devices, in: Proc. IEEE INFOCOM 2018, IEEE, 2018, pp. 46–54.
[7] X. Chen, L. Pu, L. Gao, W. Wu, D. Wu, Exploiting massive D2D collaboration for energy-efficient mobile edge computing, IEEE Wireless Commun. 24 (4) (2017) 64–71.
[8] S. Guo, B. Xiao, Y.Y. a.Y. Yang, Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing, in: Proc. IEEE INFOCOM 2016, 2016, pp. 1–9.
[9] S. Zhang, Y. Liang, J. Ge, M. Xiao, J. Wu, Provably efficient resource allocation for edge service entities using hermes, IEEE/ACM Trans. Networking 28 (4) (2020) 1684–1697.
[10] R.S. Sutton, A.G. Barto, Reinforcement learning: An introduction, MIT press, 2018.
[11] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533.
[12] J.X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J.Z. Leibo, R. Munos, C. Blundell, D. Kumaran, M. Botvinick, Learning to reinforcement learn, arXiv preprint arXiv:1611.05763 (2016).
[13] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, H. Balakrishnan, Mahimahi: accurate record-and-replay for HTTP, in: Proc. USENIX ATC 2015, 2015, pp. 417–429.
[14] L. Wei, W. Miao, Z. Zeng, H. Sun, Z. Qian, EDC: An edge-Oriented dynamic resource configuration strategy, in: Proc. Spaccs 2020, 12383, Springer, 2020, pp. 3–14.
[15] S. Sundar, B. Liang, Offloading dependent tasks with communication delay and deadline constraint, in: Proc. IEEE INFOCOM 2018, pp. 37–45.
[16] Jetson AGX Xavier, (https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-agx-xavier/, accessed on Sep 2021).
[17] V. Bharadwaj, D. Ghose, T.G. Robertazzi, Divisible load theory: a new paradigm for load scheduling in distributed systems, Cluster Comput. 6 (1) (2003) 7–17.
[18] J. Sohn, T.G. Robertazzi, S. Luryi, Optimizing computing costs using divisible load analysis, IEEE Trans. Parallel Distrib. Syst. 9 (3) (1998) 225–234.
[19] S. Zhang, J. Wu, S. Lu, Distributed workload dissemination for makespan minimization in disruption tolerant networks, IEEE Trans. Mob. Comput. 15 (7) (2016) 1661–1673.
[20] S. Zhang, J. Wu, S. Lu, Minimum makespan workload dissemination in DTNs: making full utilization of computational surplus around, in: Proc. of ACM MobiHoc 2013, pp. 293–296.
[21] J. Kim, Y. Jung, H. Yeo, J. Ye, D. Han, Neural-enhanced live streaming: improving live video ingest via online learning, in: Proc. ACM SIGCOMM, 2020, pp. 107–125.
[22] Softmax, (https://en.wikipedia.org/wiki/Softmax_function,accessed on Jul 2025).
[23] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, Continuous control with deep reinforcement learning, arXiv preprint arXiv:1509.02971 (2015).
[24] L. Wang, L. Jiao, T. He, J. Li, M. Mühlhäuser, Service entity placement for social virtual reality applications in edge computing, in: Proc. IEEE INFOCOM 2018, pp. 468–476.
[25] Y. Liang, J. Ge, S. Zhang, J. Wu, L. Pan, T. Zhang, B. Luo, Interaction-Oriented service entity placement in edge computing, IEEE Trans. Mob. Comput. 20 (3) (2019) 1064–1075.
[26] Y. Liang, J. Ge, S. Zhang, J. Wu, Z. Tang, B. Luo, A utility-Based optimization framework for edge service entity caching, IEEE Trans. Parallel Distrib. Syst. 30 (11) (2019) 2384–2395.
[27] S. Kosta, A. Aucinas, P. Hui, R. Mortier, X. Zhang, Thinkair: dynamic resource allocation and parallel execution in the cloud for mobile code offloading, in: Proc. IEEE INFOCOM 2012, pp. 945–953.
[28] M.S. Gordon, D.A. Jamshidi, S. Mahlke, Z.M. Mao, X. Chen, COMET: Code offload by migrating execution transparently, in: Proc. USENIX ODSI 2012, pp. 93–106.
[29] C. You, K. Huang, H. Chae, B.-H. Kim, Energy-Efficient Resource Allocation for Mobile-Edge Computation Offloading, arXiv preprint arXiv:1605.08518 (2016).
[30] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, P. Bahl, MAUI: Making smartphones last longer with code offload, in: Proc. ACM MobiSys 2010, pp. 49–62.
[31] C. Wang, S. Zhang, H. Zhang, Z. Qian, S. Lu, Edge cloud capacity allocation for low delay computing on mobile devices, in: Proc. IEEE ISPA 2017, pp. 290–297.
[32] L. Yang, J. Cao, H. Cheng, Y. Ji, Multi-user computation partitioning for latency sensitive mobile cloud applications, IEEE Trans. Comput. 64 (8) (2015) 2253–2266.
[33] X. Chen, L. Jiao, W. Li, X. Fu, Efficient multi-User computation offloading for mobile-Edge cloud computing, IEEE/ACM Trans. Networking 24 (5) (2016) 2795–2808.
[34] Q. Zhang, F. Liu, C. Zeng, Adaptive interference-Aware VNF placement for service-Customized 5G network slices, in: Proc. IEEE INFOCOM 2019, pp. 1–9.
[35] V.V. Vazirani, Approximation algorithms, Springer, 2004.
[36] B. Gao, Z. Zhou, F. Liu, F. Xu, Winning at the starting line: joint network selection and service placement for mobile edge computing, in: Proc. IEEE INFOCOM 2019, pp. 1–9.
[37] Z. Chen, X. Wang, Decentralized computation offloading for multi-user mobile edge computing: a deep reinforcement learning approach, EURASIP J. Wirel. Commun. Netw. 188 (2020).
[38] J. Ren, H. Wang, T. Hou, S. Zheng, C. Tang, Collaborative edge computing and caching with deep reinforcement learning decision agents, IEEE Access 8 (2020) 120604–120612.
[39] Z. Xu, L. Zhou, S.C.-K. Chau, W. Liang, Q. Xia, P. Zhou, Collaborate or separate? distributed service caching in mobile edge clouds, in: Proc. IEEE INFOCOM 2020, IEEE, 2020, pp. 2066–2075.
[40] N. Chen, S. Zhang, J. Wu, Z. Qian, S. Lu, Learning scheduling bursty requests in mobile edge computing using deepload, Comput. Networks 184 (2021) 107655.
[41] W. Zhang, Z. Zhang, H.-C. Chao, Cooperative fog computing for dealing with big data in the internet of vehicles: architecture and hierarchical resource management, IEEE Commun. Mag. 55 (12) (2017) 60–67.
[42] Y. Liang, S. Zhang, J. Wu, Volatile MAB-based configuration selection for offloading video analytics tasks to edges, in: Proc. IEEE ICASSP 2025, IEEE, 2025, pp. 1–5.
[43] T. Ouyang, X. Chen, L. Zeng, Z. Zhou, Cost-aware dispersed resource probing and offloading at the edge: a user-centric online layered learning approach, IEEE Trans. Serv. Comput. (2024).
[44] H. Xue, Y. Xia, N.N. Xiong, D. Zhang, S. Pei, Ddps: dynamic differential pricing-based edge offloading system with energy harvesting devices, IEEE Trans. Network Sci. Eng. (2025).

[45] Y. Chai, X.-J. Zeng, Q. Chen, L. Cheng, Stackelberg game-Based joint computing resource allocation and task offloading method in edge computing, IEEE Trans. Veh. Technol. (2025).
[46] Y. Liang, S. Zhang, J. Wu, Scrava: super resolution-Based bandwidth-Efficient cross-Camera video analytics, IEEE Trans. Mob. Comput. 24 (1) (2025) 293–305.
[47] M. Cheng, J. Li, P. Bogdan, S. Nazarian, H_2O-Cloud: a resource and quality of service-Aware task scheduling framework for warehouse-Scale data centers, IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 39 (10) (2020) 2925–2937. https://doi.org/10.1109/TCAD.2019.2930575
[48] M. Cheng, J. Li, S. Nazarian, DRL-Cloud: deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers, in: Proc. ASP-DAC 2018, 2018, pp. 129–134.
[49] 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. IEEE INFOCOM 2018, IEEE, 2018, pp. 1871–1879.
[50] H. Zhang, W. Li, S. Gao, X. Wang, B. Ye, Reles: a neural adaptive multipath scheduler based on deep reinforcement learning, in: Proc. IEEE INFOCOM 2019, IEEE, 2019, pp. 1648–1656.
[51] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, M. Bennis, Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning, IEEE Internet Things J. 6 (3) (2018) 4005–4018.

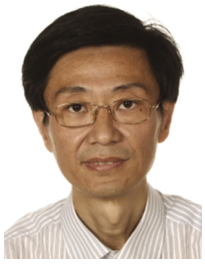
Yu Liang is an associate professor at Nanjing Normal University. She received the MS and PhD degrees from Nanjing University in 2011 and 2021, respectively. She was a senior software engineer in Trend Micro China Development Center between 2011 and 2017. Her research interests include resource allocation in cloud and edge computing. Her publications include those appeared in TON, TMC, TPDS, TON, INFOCOM, MM, ICDCS, and ICDE.

Jidong Ge is an associate professor at Software Institute, Nanjing University. He received his PhD degree in Computer Science from Nanjing University in 2007. His current research interests include cloud computing, distributed computing, workflow scheduling, workflow modeling, process mining. His research results have been published in more than 60 papers in international journals and conference proceedings including IEEE TSC, JASE, FGCS, JSS, Inf. Sci., ESA, ICSE, APSEC, ICSSP, HPCC, SEKE etc.

Jie Wu is the Director of the Center for Networked Computing and Laura H. Carnell professor at Temple University. He also serves as the Director of International Affairs at College of Science and Technology. He served as Chair of Department of Computer and Information Sciences from the summer of 2009 to the summer of 2016 and Associate Vice Provost for International Affairs from the fall of 2015 to the summer of 2017. Prior to joining Temple University, he was a program director at the National Science Foundation and was a distinguished professor at Florida Atlantic University. His current research interests include mobile computing and wireless networks, routing protocols, cloud and green computing, network trust and security, and social network applications. Dr. Wu regularly publishes in scholarly journals, conference proceedings, and books. He serves on several editorial boards, including IEEE Transactions on Mobile Computing, IEEE Transactions on Service Computing, Journal of Parallel and Distributed Computing, and Journal of Computer Science and Technology. Dr. Wu was general co-chair for IEEE MASS 2006, IEEE IPDPS 2008, IEEE ICDCS 2013, ACM MobiHoc 2014, ICPP 2016, and IEEE CNS 2016, as well as program co-chair for IEEE INFOCOM 2011 and CCF CNCC 2013. He was an IEEE Computer Society Distinguished Visitor, ACM Distinguished Speaker, and chair for the IEEE Technical Committee on Distributed Processing (TCDP). Dr. Wu is a CCF Distinguished Speaker and a Fellow of the IEEE. He is the recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award.

Sheng Zhang is an associate professor in School of Computer Science, Nanjing University. He is also a member of the State Key Lab. for Novel Software Technology. He received the BS and PhD degrees from Nanjing University in 2008 and 2014, respectively. His research interests include cloud computing and edge computing. To date, he has published more than 80 papers, including those appeared in *JSAC*, *TMC*, *TON*, *TPDS*, *TC*, *MobiHoc*, *ICDCS*, and *INFOCOM*. He received the Best Paper Award of IEEE ICCCN 2020 and the Best Paper Runner-Up Award of IEEE MASS 2012. He is the recipient of the 2020 ACM Nanjing Rising Star Award and the 2015 ACM China Doctoral Dissertation Nomination Award.

Shiwu Wen received the BS degree at the Software Institute, Xi'an Jiaotong University. He is currently working toward the MS degree at the Software Institute, Nanjing University, under the supervision of Prof. Jidong Ge. His research interests include Machine Learning and Reinforcement Learning.



Bin Luo is a full Professor at the Software Institute, Nanjing University. His main research interests include cloud computing, computer network, workflow scheduling, software engineering. His research results have been published in more than 60 papers in international journals and conference proceedings including IEEE TSC, ACM TOIST, FGCS, JSS, Inf. Sci., ESA etc. He is leading the institute of applied software engineering at Nanjing University.