Liu *et al. Journal of Cloud Computing* (2023) 12:58
https://doi.org/10.1186/s13677-023-00434-6

Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

Robust-PAC time-critical workflow offloading in edge-to-cloud continuum among heterogeneous resources

Hongyun Liu[1,2*], Ruyue Xin[1], Peng Chen[3*], Hui Gao[4], Paola Grosso[1] and Zhiming Zhao[1*]

Abstract

Edge-to-cloud continuum connects and extends the calculation from edge side via network to cloud platforms, where diverse workflows go back and forth, getting executed on scheduled calculation resources. To better utilize the calculation resources from all sides, workflow offloading problems have been investigating lately. Most works focus on optimizing constraints like: latency requirements, resource utilization rate limits, and energy consumption bounds. However, the dynamics among the offloading environment have hardly been researched, which easily results in uncertain Quality of Service(QoS) on the user side. Any part of the workload change, resource availability change or network latency could incur dynamics in an offloading environment. In this work, we propose a robust PAC (probably approximately correct) offloading algorithm to address this dynamic issue together with optimization. We train an LSTM-based sequence-to-sequence neural network to learn how to offload workflows in edge-to-cloud continuum. Comprehensive implementations and corresponding comparison against state-of-the-art methods demonstrate the robustness of our proposed algorithm. More specifically, our algorithm achieves better offloading performance regarding dynamic heterogeneous offloading environment and faster adaptation to newly changed environments than fine-tuned state-of-the-art RL-based offloading methods.

Keywords Workflow offloading, Meta learning, Time-critical, Robustness, Heterogeneous resources, MEC

Introduction

Wide use of edge-to-cloud continuum promotes a novel paradigm empowering intelligent and diverse applications in our daily life: intelligent transportation, intelligent home, and E-Healthcare. However, such a paradigm also brings new challenges: the growing computation requirements on the user side, increasing data transmission, continuous interactive computation, and communication. With this trend, task offloading is a very widely used approach to better utilize diverse computation resources both on the edge side and cloud side, which contribute to an extended calculation pipeline together-edge-to-cloud continuum. Within the popularity of the edge-to-cloud continuum, how to offload workflows properly matters in many contexts: energy consumption, latency control, and QoS. Moreover, with the evolution of the cellular network [1], the overall number of end-users

*Correspondence:
Hongyun Liu
h.liu@uva.nl
Peng Chen
chenpeng@mail.xhu.edu.cn
Zhiming Zhao
z.zhao@uva.nl
[1] Multiscale Networked Systems (MNS), University of Amsterdam, Amsterdam, The Netherlands
[2] Graduate School Informatics, University of Amsterdam, Amsterdam, The Netherlands
[3] School of Computer and Software Engineering, Xihua University, Chengdu, China
[4] College of Electrical and Control Engineering, Shaanxi University of Science and Technology, Xi'an, China

is increasing dramatically [2, 3].With the rocketing development on both sides of users and service suppliers, offloading gains importance in a more heterogeneous environment where nodes have diverse capacities. The execution becomes more complicated with more resource options. Optimization on the edge side takes many aspects into account at the same time: execution capability, execution time, which are often contradicting against each other.

To address this NP-hard problem, many works have been done [4–6]. Among them, machine learning-based approaches especially Reinforcement-Learning(RL)-based approaches have been investigated a lot: Liu, et al. [6] proposed a robust scheduling framework for independent tasks. Liu, et al. [7] proposed a multi-objective optimization framework for time-critical task scheduling. There also have been many works addressing heterogeneity in the offloading environment [8, 9]. Chen, et al. [10] propose an end-edge-cloud architecture of vehicles for task computation offloading, where considers three task computing methods. For the dynamically changing environment in the IoV, they adopt an Asynchronous Advantage Actor-Critic (A3C) based computation offloading algorithm to solve the problem and seek optimal offloading decisions. As workflows consist of tasks and their dependencies, when the tasks come with time-critical constraints the workflows also need to take these constraints into account. Chen, et al. [11] develop a distributed multihop task offloading decision model for task execution efficiency, which consists of two parts: 1) a candidate vehicle selection mechanism for screening the neighboring vehicles that can participate in offloading and 2) a task offloading decision algorithm for obtaining the task offloading solution. Wei, et al. [12] improve the nondominated sorting genetic algorithm II (NSGA-II) by modifying the initial population according to the matching factor, dynamic crossover probability and mutation probability to promote excellent individuals and increase population diversity. Therefore, when we optimize offloading policies, we also need to meet the time-critical or latency requirements of those workflows [13, 14].

However, after reviewing related papers and work done lately, we find that the robustness of the offloading performance has rarely been addressed in a dynamic heterogeneous resource edge-to-cloud continuum environment. The robustness of offloading performance refers to the stability of the offloading performance in a dynamic environment, regarding performance measurements. The absence of robustness results in offloading performance deviation, which brings in the uncertainties to latency. Furthermore, the uncertain latency influences the QoS even end up in violation of Service Legal Agreement(SLA). In our work, we propose a Meta-PAC(probably approximately correct)-Reinforcement-Learning-based robust offloading algorithm(MLR-LC-DRLO) to address this issue in a heterogeneous environment. The main contributions of this paper include:

1 Workflow offloading in the heterogeneous environment: we build up a heterogeneous environment to investigate workflow offloading.
2 Time-critical workflow offloading: we design a PAC Reinforcement-Learning scheme to learn offloading policy. The learning process is with maximum exploration limit, which is based on workflow latency. In this way it offers offloading latency guarantee and makes the learning process more efficient.
3 Robust workflow offloading: we propose a Meta-Learning-based offloading algorithm, achieving more robust offloading performance compared with typical RL-based offloading approaches.

In the remainder of this paper, firstly we give the general formulation of the offloading in Problem formulation section. Followed by Related work section, where we go through the related work. Then we propose the detailed framework and algorithm MLR-LC-DRLO in Methodology section. Next, we evaluate the robustness performance and optimization performance with comprehensive implementations in Evaluation section. We further discuss the implementation results and make future work plans in Discussion section. Finally, Conclusion section summarizes the whole paper.

Problem formulation

We formulate offloading in a typical use case, as shown in Fig. 2, the workflow including the requests and corresponding dependencies firstly go to the local scheduler. After local scheduler makes the decision whether to calculate the request locally or offload them to MEC host. Between MEC host and end users, there is the MEC network connecting the two parts, including the up link and down link. Then if the decision is to offload the request to the MEC host, the request will be transmitted to the MEC host, where the offloading orchastrator will allocate them to different VMs through gateways. In this work, the resource composition of each VM on MEC host side is heterogeneous.

After we present the typical offloading pipeline, we formulate each part of the pipeline step by step. First of all, it's the workflow model. As is known, workflows consist of tasks and their dependencies. Here we define the workflow model as $\mathcal{D} = (TA, \overrightarrow{ED})$, where we use TA to represent the tasks set, based on this we use the vector $\overrightarrow{ED}$ represents the dependencies, which are described as directed

edge connected between the tasks respectively. We take $\overrightarrow{ed} = (ta_i, ta_j)$ as an example, where $\overrightarrow{ed}$ denotes the dependency between task ta_i and task ta_j meaning ta_j is an immediate successor task of ta_i. We also formulate several principles of workflow models as follows:

1. As is shown in Fig. 1, for the two connected tasks, for example A and B, the one starts its execution earlier (A) is the leading task, the other one (B) is successor task.
2. The execution of a successor task only starts later than the ending of its leading task's execution until the last one.
3. The tasks have no successor tasks are the exit tasks.

For different use cases and applications, the VMs and containers are getting more diverse, that is where the offloading heterogeneity comes. Based on the formulation of the workload, the heterogeneity of the environment comes from the heterogeneous resource composition of each VMs. Here we define ξ type of VMs, their computation capacities are represented as $Cap_l, l \in [1, 2, 3, ..., \xi]$. For each task ta_i, it has several information including: the resource requirement for running task, Cp_i, the sent data sizes, Da_i^s, and the received result data size , Da_i^r. After we formulate the tasks model and the VMs, we turn to the MEC model, which consists of: the wireless up-link channel transmission rate, UT, and the down-link channel transmission rate DT. Based on this formulation, the latency of task ta_i sending data, Lat_i^{UI}, is calculated as:

$$Lat_i^{UI} = Da_i^s / UT \tag{1}$$

getting executed on the MEC host, Ex_i^s, is calculated as:

$$Ex_i^s = Cp_i / Cap_l \tag{2}$$

receiving the result data, Lat_i^D, is calculated as:

$$Lat_i^D = Da_i^r / DT \tag{3}$$

When a task ta_i gets scheduled to be executed locally, the latency is just the time spent on local execution on the end-user side, which is calculated as

$$Lat_i^{Lo} = Cp_i / Cap_{Lo} \tag{4}$$

where Cap_{Lo} represents the computational capacity of the end-user.

Once a task ta_i gets offloaded to the MEC host, the total latency are the sum of latency from all parts, which includes local processing, up-link transmission, and remote processing latency and results transmission latency, as shown in Fig. 2. Based on the aforementioned model, we further formulate the offloading policy into $Pol_{1:n} = a_1, a_2, ..., a_n$, where a_i represents the corresponding offloading decision of each ta_i.

The finishing time of the process on the up-link channel, $\mathcal{T}_i^{UI}$, are defined as:

$$\mathcal{T}_i^{UI} = max\{Av_i^{UI}, \max_{j \in parent(t_i)} \{\mathcal{T}_j^{UIE}, \mathcal{T}_j^D\}\} + Lat_i^{UI}, Av_i^{UI} = max\{Av_{i-1}^{UI}, \mathcal{T}_{i-1}^{UI}\} \tag{5}$$

The finishing time of ta_i's execution on the MEC host, FT_i^s, and finishing time of its process on the down-link channel, FT_i^D are defined as:

$$\begin{aligned} \mathcal{T}_i^s &= max\{Av_i^s, max\{\mathcal{T}_i^{UI}, \max_{j \in parent(t_i)} \{\mathcal{T}_j^s\}\}\} + Lat_i^{UI}, \\ Av_i^s &= max\{Av_{i-1}^s, \mathcal{T}_{i-1}^s\}, \\ \mathcal{T}_i^D &= max\{Av_i^D, \mathcal{T}_i^s\} + Lat_i^D, Av_i^D = max\{Av_{i-1}^D, \mathcal{T}_{i-1}^D\}. \end{aligned} \tag{6}$$

The completion time of task ta_i on the end user side, FT_i^{UIE}, are defined as:

$$\mathcal{T}_i^{UIE} = max\{Av_i^{UIE}, \max_{j \in parent(t_i)} \{\mathcal{T}_j^{UIE}, \mathcal{T}_j^D\}\} + Lat_i^{UIE} \tag{7}$$

$$Av_i^{UIE} = max\{Av_{i-1}^{UIE}, \mathcal{T}_{i-1}^{UIE}\}. \tag{8}$$

Overall, given a offloading policy model $Pol_{1:n}$, the total latency of a DAG, $Lat_{A_{1:n}}^c$, is defined as:

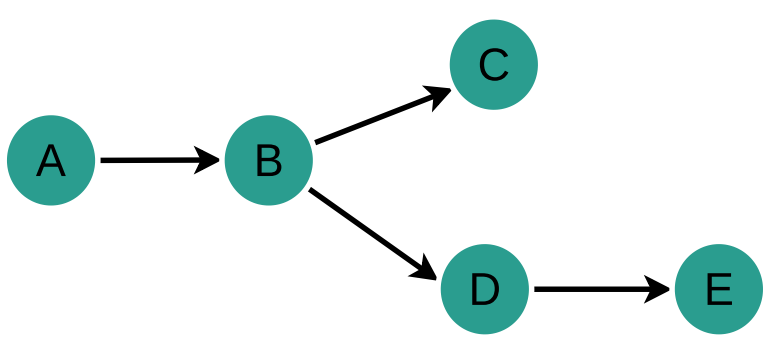

Fig. 1 Workflow Model

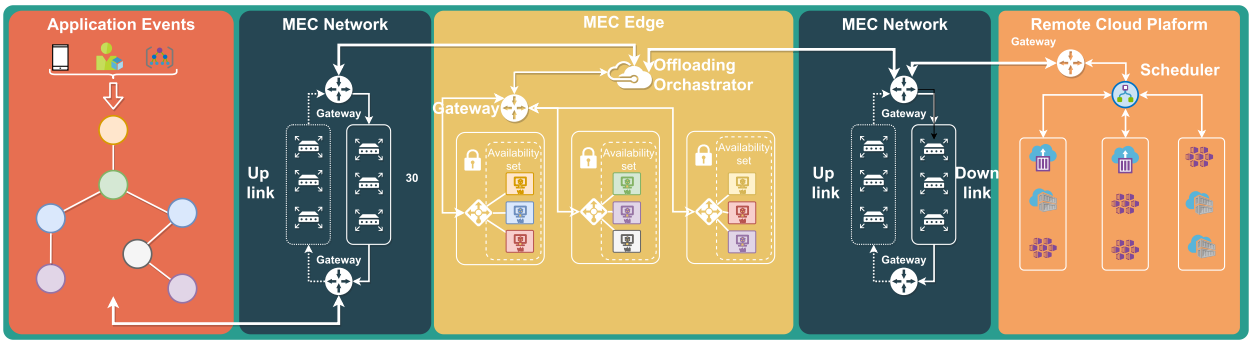

Fig. 2 Overall Offloading Process

$$Lat^{c}_{Pol_{1:n}} = max[\max_{t_k \in \mathcal{K}}\{\mathcal{T}^{UE}_k, \mathcal{T}^{D}_k\}] \tag{9}$$

where $\mathcal{K}$ denotes the exit tasks set, which consists of the tasks which have no successor tasks. In the next section, we will propose the detailed offloading algorithm based on the model formulation (Table 1).

Related work

Learning-Based Offloading Han, et al. [15] proposes a deep reinforcement learning-based approach to offloading decision-making in mobile edge computing. Min, et al. [16]proposed a deep RL-based offloading enabling the IoT device to optimize the offloading policy without knowledge of the MEC model, the energy consumption model, and the computation latency model. Dinh, et al. [17] proposed a model-free reinforcement learning offloading mechanism which helps MUs learn their long-term offloading strategies to maximize their long-term utilities. Cheng, et al [18] propose a deep reinforcement learning-based computing offloading approach to learn the optimal offloading policy on-the-fly, where we adopt the policy gradient method to handle the large action space and actor-critic method to accelerate the learning process. Some work also adopted LSTM network the to do prediction of the environment state [19]. Meta-Learning has also been investigated [20] to offer an fast adaptive offloading method-MRLCO. Cao et al. proposed a novel multi-agent DRL based approach [21], which adopts act-critic neural networks to calculate Q-value based on corresponding reward function. DPM framework proposed by [22] applied the long short-term memory (LSTM) neural network investigated the prediction and strategies of resource allocation under the objective of energy consumption reduction in cloud-edge continuum.

Some work also adopted LSTM network the to do prediction of the environment state [19]. Meta-Learning has also been investigated [20] to offer an fast adaptive offloading method-MRLCO. Cao et al. proposed a novel

Table 1 Notation Summary

Symbols	Explanation
$\mathbb{E}()$	Mean value calculation or function
ta_i	task i
Da^s_i	Size of data sent by task ta_i
Da^r_i	Size of data received by task ta_i
UT, DT	Transmission rate of up-link
DT	Transmission rate of down-link
Cap_{Lo}	Computational capacity of UE
Cap_l	Computational capacity of VM l
$Lat^{ul}_i, Lat^s_i, Lat^{dl}_i, Lat^{UE}_i$	Latency of task ta_i from up-link channel, from MEC host side, from down-link channel, and from UE respectively.
$\mathcal{T}^U_i, \mathcal{T}^s_i, \mathcal{T}^D_i, \mathcal{T}^{UE}_i$	Finishing time of task ta_i on up-link channel, MEC host, down-link channel, and UE
$Av^U_i, Av^s_i, Av^D_i, Av^{UE}_i$	For specific task ta_i, the available time of up-link channel, MEC host, down-link channel, and UE respectively
$Pol_{1:n}$	Offloading policies for task set including $ta_i...ta_n$
$\mathcal{T}_i, \rho(\mathcal{T})$	A learning task and distribution of learning tasks
s_i, a_i, r_i	the i-th state, i-th action, and i-th reward of an MDP
$\pi(a\|s;\theta)$	Offloading policy model
$v(s;\theta)$	Value function
τ_π	Trajectories sampled via policy model π.
$\mathcal{F}_{en}, \mathcal{F}_{de}$	Encoder functions and decoder function
e_i, d_i	Encoder output and decoder output at time step i
c_i	Context vector at decoding step i
$\hat{A}_t$	Advantage function value
$Up(\theta, \mathcal{T}_i)$	Learning optimizer function (e.g., Adam)

multi-agent DRL based approach [21], which adopts act-critic neural networks to calculate Q-value based on corresponding reward function. Shan et al. integrated DRL and Federated Learning to optimize resource allocation problems, which offers acceleration of DRL agents training. Lolos et al. proposed a novel full-model based RL [23] for elastic resource management, employs adaptive state space partitioning.

Resource Heterougeneity Guan, et al. [24] propose a novel hybrid offloading model to solve the heterogeneous resource-constraint offloading issues in the Cloudlet, concerning the offloading energy and execution efficiency. Li, et al. [25] propose a task offloading strategy in the MEC system with a heterogeneous edge by considering the execution and transmission of tasks under the task offloading strategy, we present an architecture for the MEC system. Xiong, et al. [26] propose an intelligent task offloading framework in heterogeneous vehicular networks with three Vehicle-to-Everything (V2X) communication technologies, namely Dedicated Short Range Communication (DSRC), cellular-based V2X (C-V2X) communication, and millimeter wave (mmWave) communication.However, with the growing attention paid to offloading issues, there are still several issues missing among them: the absence of the accurate *robust* solution when the dynamics occur in the offloading environment; the absence of the recovery *robust* solution after the performance deviation brought by the dynamics. During the past ten to twenty years, cloud-edge continuum has been further investigated, many new topics attract attention. Among those topics offloading, as an essential part of cloud-edge continuum, has been studied [27]. There has been many offloading solutions have been investigated and proposed from different perspectives: using hierarchical method [28], or collaborative optimization method [29], energy-efficient method [30]. The optimization performance of the conventional approaches often come from explicit models based on different resources or workflows and corresponding offloading policies models sometimes even a very specific system. As with the increasing popularity, Machine Learning-based optimization solutions also have hence attracted certain research attention [4, 5] in context of offloading. Among Machine Learning-based approaches, Reinforcement Learning-based approaches [5, 6, 17, 31] optimize offloading interactively without asking for data labelling. However, the performance of the approaches aforementioned is depending on and easily influenced by the dynamics from each component of the MEC pipeline: the resource availability, the request pattern, the data transmission latency. Thus, any changes from those parts could lead to performance deviation for those approaches, which asks for repeating of the pruning process or training process when it comes to learning-based solutions. From the robustness perspective, the higher deviation means the lower robustness of the offloading performance. There are some work addressing this issue from robustness perspective: adaptive optimization approach [20], connection stability [32], robust network contention [33]. However, compared with throughput or energy consumption, the offloading robustness among heterogeneous resources environments has not been well addressed lately. In the next section, we will formulate our approach step by step.

PAC-RL: Fiechter [34] first proposed the PAC RL framework, and algorithms with sample complexity O((SAH3/2) log(1/)) have been developed [35, 36], which are minimax-optimal in time-inhomogeneous MDPs [37]. These algorithms combine a well-chosen halting rule with an optimistic sampling rule. Most optimistic sampling strategies have been presented for regret minimization, where the policy t is the greedy policy with regard to an upper confidence constraint on the optimal Q function. In specifically, episodic MDPs are reached via the UCBVI method of Azar et al. [38] (with Bernstein bonuses). Instance-dependent upper limits on the regret for optimistic algorithms have been presented in recent publications [39–41]. A complexity term that is dependent on the MDP instance is present in an instance-dependent bound, generally through the idea of a sub-optimality gap. In particular, Wagenmaker et al. [42] shown that optimistic no-regret sampling procedures cannot attain the instance-optimal rate for PAC identification. The basic idea is that an ideal PAC RL algorithm must visit each state-action combination at least a specific number of times, necessitating the use of playing strategies that cover the whole MDP in the fewest possible episodes. A regret-minimizer, on the other hand, concentrates on using high-reward strategies that, depending on the MDP instance, may be arbitrarily ineffective in traveling to remote states.

Methodology

In this section we elaborate the approach we propose: MLR-LC-DRLO in details. We firstly start with the formulation of latency-critical PAC-RL:

Latency-critical Probably Approximate Correct (PAC) reinforcement learning

With the conventional Reinforcement Learning set up, there is rare upper bound of offloading accuracy during the exploration process, which leads the optimization to undesired directions, wasting training time. So here we firstly formulate this upper bound of offloading to limit the training time and accuracy more preciscely. When there exit dynamics in the training environment, every time during the transition after the dynamic disturbances, the learning

process needs to optimize the offloading policies from the scratch again. During this process, specific upper bound on the exploration of training will save training time and offer better accuracy. And also guidance during the transition process could also save retraining time. To this end, we propose a probably approximate correct RL-based offloading algorithm, which offers upper bound on the exploration process:

$$\tilde{O}(n_S^2 \times n_A/(\varepsilon^3(1-\gamma)^6)) \tag{10}$$

where, n_S denotes number of states, n_S represents the number of actions, ε is the accuracy parameter, and γ is the discount factor. The proof follows. In our latency-critical PAC reinforcement-learning formulation, we take $\mathcal{M}$ as a finite Markov Decision Process(MDP), denoted as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Gamma)$. Within $\mathcal{M}$, we take: $\mathcal{S}$ as the states set, $\mathcal{A}$ as the actions sets corresponding to each state, $\mathcal{T}$ as the transition distribution and represented as: $\mathcal{S} \times \mathcal{A} \rightarrow \Lambda_{\mathcal{S}}$, $\mathcal{R}$ is the reward distribution, and $r \in [0, 1)$ is a reward discount factor. $\mathcal{T}(s^{'}|s, a)$ indicates the probability of the transition from states s to state $s^{'}$ out of the distribution $\mathcal{T}(s, a)$. Each time-step here is defined as a single time interaction between the learner and the environment. Each time interaction between learning agent and the environment is described as a state-action pair (s, a) including the information of that the learner takes the specific action a from the state s. We use $R(s, a)$ to denote the expected reward out of reward distribution $\mathcal{R}(s, a)$. During the Learning process, the learner accumulates the rewards $r \sim \mathcal{R}(s, a)$ when takes each action a at state s then transits to next state $s^{'}$ with the possibility: $s^{'} \sim \mathcal{T}(s, a)$. By repeating this process, the objective of the learner tries to achieve the objective, which is accumulating possible most reward within possible least times of attempts. A policy set consists of any strategy followed by the learner choosing actions. A stationary policy refers to the policy that produces an action based on only the current state, without considering the previous interaction experiences. For policy π, the discounted, infinite-horizon value function from state s is formulated as follows:

$$V_{\mathcal{M}}^{\pi}(s) = \mathrm{E}[\sum_{\mathrm{j}=1}^{\infty} \mathbf{r}^{\mathrm{j}-1} \mathrm{r_j}|\mathrm{s}] \tag{11}$$

$$Q_{\mathcal{M}}^{\pi}(s) = \mathrm{E}[\sum_{\mathrm{j}=1}^{\infty} \mathbf{r}^{\mathrm{j}-1} \mathrm{r_j}|\mathrm{s}] \tag{12}$$

where, H represents the number of the steps, which is a positive integer, $V_{\mathcal{M}}^{\pi}(s, H)$ indicates the accumulated value out of H-step under policy π, starting from state s. Specifically, let s_t and ∇_t be the tth encountered state and received reward, respectively, resulting from execution of policy π in MDP $\mathcal{M}$. Here we define policy model π as non-stationary considering the dependencies among tasks. Here we define $c = (s_1, a_1, r_1, s_2, a_2, r_2, ...)$ as a learning path of $\mathcal{A}$. In this manner, at time t the state s_t is described as a serial state-action experiences denoted as: $c_t = (s_1, a_1, \mathrm{r}_1, ..., \mathrm{s_t})$. Then we derive the expected value functions as follows:

$$V_{\mathcal{M}}^{\pi}(c_t) = \mathrm{E}[\sum_{\mathrm{j}=0}^{\infty} \mathbf{r}^{\mathrm{j}} \mathrm{r_{t+j}}|\mathrm{c_t}] \tag{13}$$

$$V_{\mathcal{M}}^{\pi}(c_t, H) = \mathrm{E}[\sum_{\mathrm{j}=0}^{\mathrm{H}-1} \mathbf{r}^{\mathrm{j}} \mathrm{r_{t+j}}|\mathrm{c_t}] \tag{14}$$

where the expected values take all previous possible policy paths the learner follows. The optimal policy is denoted as π^* and has value functions $V_{\mathcal{M}}^{*}(s)$ and $Q_{\mathcal{M}}^{*}(s, a)$.

Based on the primary definitions, we further define several properties used in PAC-MDP set up:

Definition of Sample Complexity of Exploration(Kakade 2003) Given an MDP $\mathcal{M}$, an learning algorithm $\mathcal{A}$ within $\mathcal{M}$, for any fixed $\varepsilon > 0$, the sample complexity of exploration of $\mathcal{A}$ is the number of timesteps t such that the policy at time t, $\mathcal{A}_t$, satisfies:

$$V^{\mathcal{A}_t}(S_t) < V^*(S_t) - \varepsilon \tag{15}$$

Definition of Efficient PAC-MDP Given an MDP $\mathcal{M}$ (here we refer the MDP we formulate as aforementioned), an learning algorithm $\mathcal{A}$ within $\mathcal{M}$, $\mathcal{A}$ is an **efficient PAC-MDP** (Probably Approximately Correct in Markov Decision Processes) algorithm when, given $\varepsilon > 0$ and $0 < \sigma < 0$, $\mathcal{A}$ satisfies: the per-timestep computational complexity, space complexity, and the sample complexity of $\mathcal{A}$ are less than some polynomial of $(S, A, 1/\varepsilon, 1/\sigma, 1/(1-\gamma))$, with probability greater than $1-\sigma$. $\mathcal{A}$ is **PAC-MDP** when the definition is relaxed to be without computational complexity requirement.

Definition of Admissible Heuristics Given an MDP $\mathcal{M}$, an learning algorithm $\mathcal{A}$ within $\mathcal{M}$, we define a function:

$$U : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R} \tag{16}$$

it is admissible heuristic when it satisfies:

$$U(s, a) \geq Q^*(s, a) \tag{17}$$

for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$.

We also assume that $U(s, a) \leq V_{max}$ for all $(s, a) \in \mathcal{S} \times \mathcal{A}$ and some quantity V_{max}. We set:

$$U(s,a) = V_{max} = 1/(1-\Gamma) \tag{18}$$

since we have: $V^*(s) = max_{a\in\mathcal{A}} Q^*(s,a)$, which is at most $1/(1-\Gamma)$. Therefore, without loss of generality, we assume

$$0 \le U(s,a) \le V_{max} \le 1/(1-\Gamma) \tag{19}$$

for all $(s,a) \in \mathcal{S} \times \mathcal{A}$.

We assume that after each time disturbance of the dynamics, before the new convergence of the training, the offloading policy is an admissible heuristic. Considering that the learner has acted with respect to some experienced state-action pair (s, a). We define $n(s, a)$ as the n-step experiences, where the learner takes action a from state s. Throughout the experiences, the received rewards at state s by taking action a: r[1], r[2], ..., r[n(s, a)]. Then, the empirical mean reward is:

$$\hat{\mathrm{R}}(s,a) := \frac{1}{n(s,a)} \sum_{i=1}^{n(s,a)} \mathrm{r[i]} \tag{20}$$

After taking an action, the learner changes the environment accordingly through this interaction. We describe this process as: the learner has taken action a from state s and immediately transitioned to the state $s^{'}$ through $n(s,a,s^{'})$ times action-taking. Throughout this process, the empirical transition distribution $\hat{T}(s,a)$ satisfies:

$$\hat{T}(s^{'}|s,a) := \frac{n(s,a,s^{'})}{n(s,a)}, \text{ for each } s^{'} \in \mathcal{S} \tag{21}$$

The objective of the learner through the learning process is to maximize the current action value, $Q(s,\cdot)$ by choosing the specific actions, offloading strategies here, and applying them to the environment. The update step is to solve the following set of Bellman equations:

$$\begin{cases} Q(s,,a) = \hat{R}(s,a) + \Gamma \Sigma_{s'} \hat{T}(s^{'}|s,a) \max_{a^{'}} Q(s^{'},a^{'}), \\ \text{if } n(s,a) \ge m_{n(s,a)}, \\ Q(s,,a) = U(s,a), \\ \text{otherwise}, \end{cases} \tag{22}$$

where $\hat{R}(s,a)$ denotes the maximum-likelihood estimates for the reward, $\hat{T}(\cdot|s,a)$ indicates transition distribution of state-action pair (s,a). That is, the computation of $\hat{R}(s,a)$ and $\hat{T}(s^{'}|s,a)$ in Eq. 22, uses only the first $n(s,a) = m$ samples. $\hat{R}(s,a)$ and $\hat{T}(\cdot|s,a)$ here are the first m times observations of (s, a). So during the transition process, instead of modeling each state-action pair, we assert their value to be $U(s, a)$. $U(s, a)$ here is guaranteed to be an upper bound on the true value function as we formulated aforementioned. To simplify the notation, we redefine $n(s, a)$ to be minimum of m and number of times state-action pair (s, a) has been experienced.

Proof

Let $Q_i(s,a)$ denote the action-value estimates after the ith iteration of value iteration. We also have:

$$\xi_i := max_{(s,a)} |Q^*(s,a) - Q_i(s,a)| \tag{23}$$

Then we have:

$$\begin{aligned} \xi_i &= max_{(s,a)} |(R(s,a) + \gamma \sum_{s'} T(s,a,s^{'}) V^*(s^{'})) \\ &\quad - (R(s,a) + \gamma \sum_{s'} T(s,a,s^{'}) V_{i-1}(s^{'}))| \\ &= max_{(s,a)} |\gamma \sum_{s'} T(s,a,s^{'})(V^*(s^{'}) - V_{i-1}(s^{'}))| \\ &\le \gamma \xi_{i-1} \end{aligned} \tag{24}$$

□

By deriving from the fact: $\xi_0 \le 1/(1-\gamma)$ we get that: $\xi_i \le \gamma^i/(1-\gamma)$. Setting this value to be at most β and solving for i yields $i \ge \frac{ln(\beta(1-\gamma))}{ln\gamma}$. We claim that:

$$\frac{ln\frac{1}{\beta(1-\gamma)}}{1-\gamma} \ge \frac{ln(\beta(1-\gamma))}{ln\gamma} \tag{25}$$

Note that (25) is equivalent to the statement $1-\gamma \le -ln\gamma$, which follows from the identity $e^x \ge 1+x$. Given the previous setup and assumption, as efficient PAC-RL, to achieve an $\alpha-$optional policy it is sufficient to run it for iterations number:

$$O\left(\frac{ln(1/(\alpha(1-\Gamma)))}{1-\Gamma}\right) \tag{26}$$

The real-valued parameter, ε_1, that specifies the desired closeness to optimality of the policies produced by value iteration. Based on this, we drive m and ε_1 with the characterization of other parameters including: $\varepsilon, \sigma, S, A, \gamma$ in context of the theoretical guarantees about the learning efficiency.

Firstly we give explicitly definition of m and ε_1 during the learning process and some internal parameters:

1 $\varepsilon_1 \in (0,1)$ is a constant added to value estimate as a bonus value of exploration.
2 m is the number of experiences of a state-action pair before performing an update.
3 $l(s, a)$ denotes the number of samples collected for (s, a).

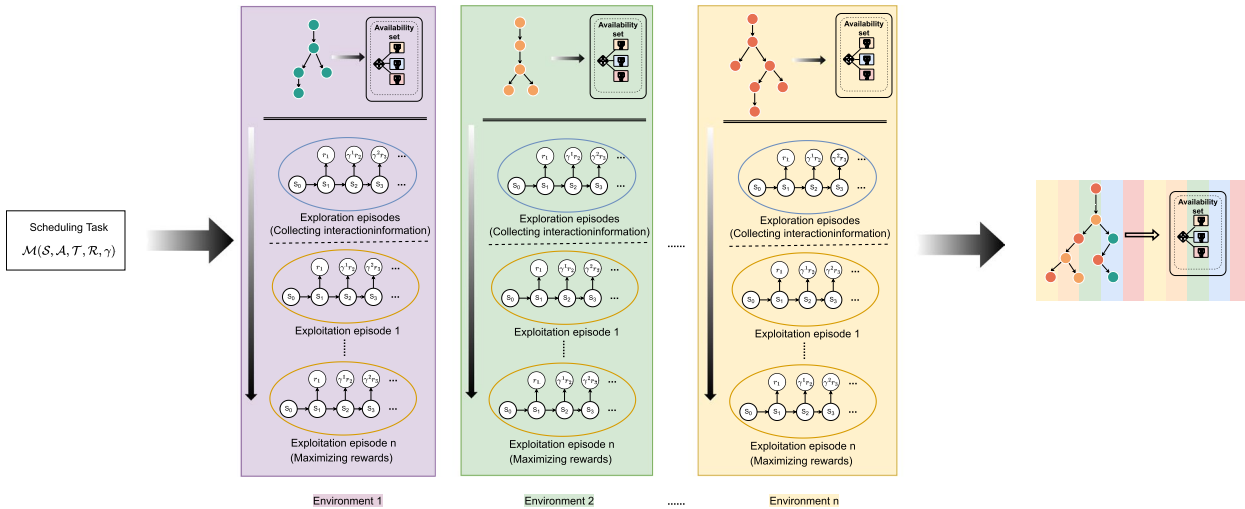

Fig. 3 Overall Learning Process

4 $AU(s, a)$ represents the the running sum of target values used to update $Q(s, a)$ once the learning agent collects enough samples.
5 $b(s, a)$ denotes the first timestep for which the first experience of (s, a) gets collected for the latest ongoing update attempt.
6 $FLG(s, a) \in \{0, 1\}$ indicated the binary value of sampling action: 1, to collect sample for (s, a); 0, not to collect sample for (s, a).

Update Rules Formulation At time t, after collecting latest m steps of experiences pairs, including next states $(s_{k_1}, s_{k_2}, ..., s_{k_m})$ in order of $k_1 < k_2 < ... < k_m$, where $k_m = t$. The received i^{th} reward is denoted as r_i. Thus, we could describe the update rule of learning agent taking action a from state s at time k_i as follows:

$$Q_{t+1}(s, a) = \frac{1}{m}\sum_{i=1}^{m}(r_{k_i} + \varepsilon V_{k_i}(r_{s_i})) + \varepsilon_1 \tag{27}$$

the condition of a an update is performed is the following equation holds:

$$Q_t(s, a) - (\frac{1}{m}\sum_{i=1}^{m}(r_{k_i} + \varepsilon V_{k_i}(r_{s_i}))) \geq 2\varepsilon_1 \tag{28}$$

Then to simplify the calculation, the learning agent only calculates the updates when the $FLG(s, a)$ is 1 (true), decreasing the update attempts to finite times. The conditions of turning $FLG(s, a)$ to be true are: firstly, initialization set up. Secondly, when any state-action pair is updated. Conditions of turning $FLG(s, a)$ from true to false is when no updates are made during a length of time for which (s, a) is experienced m times and the next attempted update of (s, a) fails. In this way no more attempted updates of (s, a) are allowed until another action-value estimate is updated.

As shown in Fig. 3, we describe the overall learning process step by step. In general, the learning agent samples m steps in different environments for exploration then turn to the exploitation process. After finishing the learning process within each environment, the learning agent turns to another environment, repeating the same learning period. Once the dynamics appear, the learning agent also sample just first m samples in the new environment, doing the exploration and exploitation with the upper bound $\tilde{O}(n_S^2 \times n_A/(\varepsilon^3(1-\gamma)^6))$. In this way, the learning agent is able to keep the learning process always with the upper bound. Especially during the process right after the dynamics, the fixed sampling complexity and exploration upper bound helps against the influences from the newly changed environment.

Formulation of latency-critical PAC-RL

In this section, we continue the formulation of the learning process one step further to the formulation of the Reinforcement Learning and the Meta Learning. Based on the MDP $\mathcal{M}$ aforementioned, we formulate the RL part as follows:

1 **State**: The needed state information of a task, ta_i, during the offloading process includes the encoded DAG dependencies and the corresponding offloading plans. The detailed state definition is as follows:

$$\mathcal{S} := \{s_i | s_i = (\mathcal{D} = (\overrightarrow{TA}, \overrightarrow{ED}), Pol_{1:i})\},\ i \in [1, |\overrightarrow{TA}|], \tag{29}$$

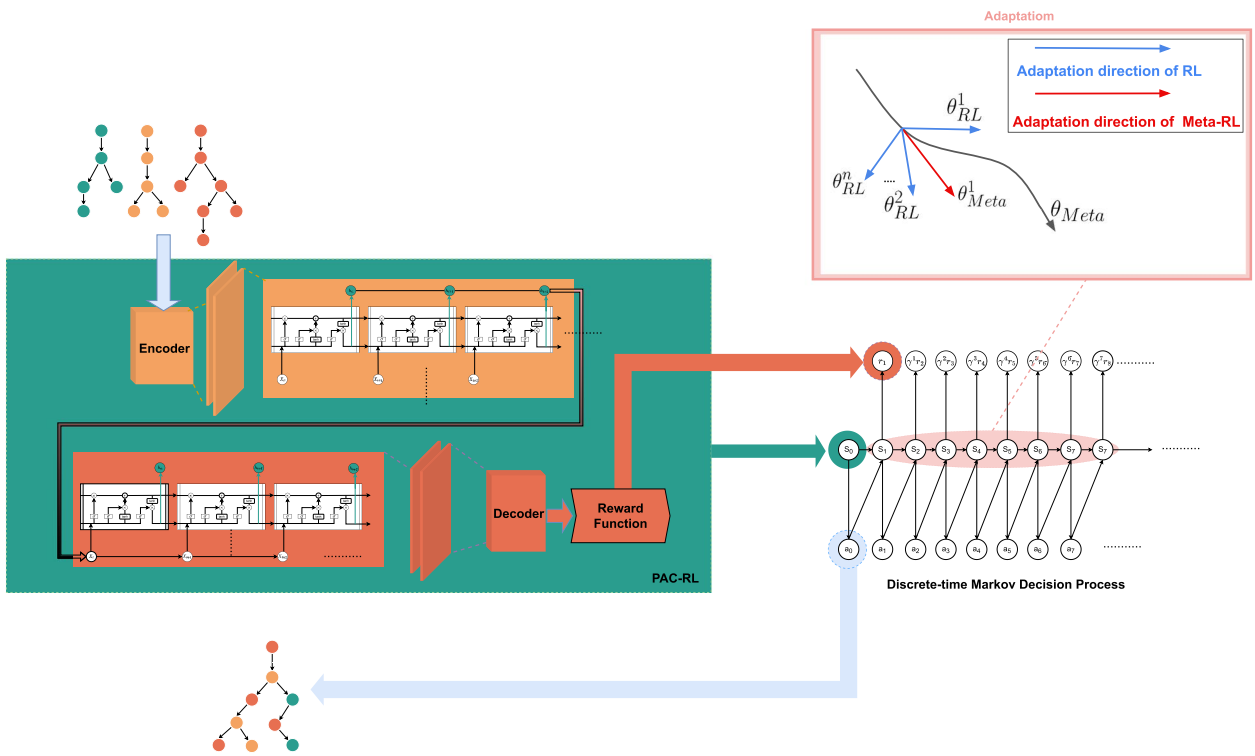

Fig. 4 Paradigm of MLR-LC-DRLO

where $\mathcal{D} = (\overrightarrow{TA}, \overrightarrow{ED})$ is a sequence of task embedding and $Pol_{1:i}$ is the offloading policy of the tasks scheduled before ta_i. Based on the above definition and formulation, we definite the offloading policy of ta_i: $Pol(a_i|\mathcal{D} = (\overrightarrow{TA}, \overrightarrow{ED}), A_{1:i-1})$ as follows:

$$\begin{aligned} &Pol(A_{1:n}|\mathcal{D} = (\overrightarrow{TA}, \overrightarrow{ED})) \\ &= \sum_{i=1}^{n} Pol(a_i|\mathcal{D} = (\overrightarrow{TA}, \overrightarrow{ED}), A_{1:i-1}) \end{aligned} \tag{30}$$

2 **Action**: The offloading choice of each task is a constant value, which indicates: execution locally, execution on different VMs with different resources. By adding up the actions of all the tasks we get the action space $\mathcal{A}$.

3 **Reward**: Throughout the learning process of offloading, minimizing latency $Lat_{A^c_{1:n}}$, defined in Eq. 9 is the primary objective. To achieve this, we formulate the reward function into an estimated negative increment of the latency calculated every execution of an offloading decision taken for a task. The detailed definition is as follows:

$$\Delta Lat^c_i = Lat_{A^c_{1:i}} - Lat_{A^c_{1:i-1}} \tag{31}$$

More detailed offloading policy model learning paradigm with aforementioned three parts is shown in Fig. 4. In our proposed training paradigm, we build up both encoder and decoder based on recurrent neural networks(RNN) [43] to learn the dependencies among tasks. First we apply the tasks embedding, which is the input of the encoder. We define $\mathcal{F}_{en}$ as the encoding function, the each step output of the encoder, e_i, is correspondingly formulated as:

$$e_i = \mathcal{F}_{en}(ta_i, e_{i-1}) \tag{32}$$

To make sure decoder learn from different part of the source sequence without information loss, we apply the attention mechanism [44]. The output of the encoder is the input of the decoder, where we define the decoding function as $\mathcal{F}_{de}$. After decoder we get the offloading policies for the workflows,d_j. The decoding process is as follows:

$$d_j = \mathcal{F}_{de}(d_{j-1}, a_{j-1}, c_j) \tag{33}$$

where c_j is the context vector at decoding step j and is computed as a weighted sum of the encoder as follows:.

$$c_j = \sum_{i=0}^{n} \alpha_{ji} e_i \tag{34}$$

The weight α_{ji} of each output of encoder, e_i is computed by

$$\alpha_{ji} = \frac{exp(f(d_{j-1}, e_i))}{\sum_{k=1}^{n} exp(f(d_{j-1}, e_i))}, \tag{35}$$

where $f(d_{j-1}, e_i)$, is used to calculate the percentage that how much possibility the input at position i matches the output at position j. Regrading the structure of NN(Neural Network), we adopt the sequence-to-sequence neural network [45], which is good at learning context information. The policy learned by NN is formulated as $Pol(a_j|s_j)$. The value function is formulated as $v_{Pol}(s_j)$. The action a_j is determined based on the following calculation:

$$a_j = argmax_{a_j} v_{Pol}(a_j|s_j) \tag{36}$$

Formulation of MLR-LC-DRLO

Based on the aforementioned PAC-RL formulation, we then optimize robustness concern by integrating Meta-Learning optimization part [46]. As to Meta-Learning optimization part, we have two loops of training: *inner loop* and *outer loop*, which we will elaborate in the following part. Overall we define the objective function based on Proximal Policy Optimization (PPO) [47]:

$$J^{C}_{ta_i}(\theta_i) = \mathbb{E}_{\tau \in P_{ta_i}(\tau, \theta^o_i)} \left[\sum_{t=1}^{n} min(Pr_t, \hat{A}_t, slice^{1+\epsilon}_{1-\epsilon}(Pr_t)\hat{A}_t) \right] \tag{37}$$

where, $\pi_{\theta^o_i}$ is the sample policy, θ^o_i is the vector of parameters of the sample policy network, π_{θ_i} is the target policy, where θ_i equals to θ^o_i at the initial epoch. Pr_t is the probability ratio between the sample policy and target policy, which is defined as

$$Pr_t = \frac{\pi_{\theta_i}(a_t|\mathcal{D}(TA, ED), A_{1:t})}{\pi_{\theta^o_i}(a_t|\mathcal{D}(TA, ED), A_{1:t})} \tag{38}$$

We also define a function $slice^{1+\epsilon}_{1-\epsilon}(Pr_t)$ to remove the incentive for moving Pr_t outside the interval $[1-\epsilon, 1+\epsilon]$ giving specific limit to the value of Pr_t.

We formulate our advantage fucntino based on general advantage estimator (GAE) [48]. The detailed formulation which is as follows:

$$\hat{A}_t = \sum_{k=0}^{n-t+1} \Big(\gamma\lambda)^k (r_{l+k} + \gamma v_\pi(s_{t+k+1}) - v_\pi(s_{t+k}) \Big), \tag{39}$$

where $\hat{A}_t$ denotes the advantage function value at time step t, $\lambda \in [0, 1]$ is used to control the trade-off between bias and variance.

Overall, we define the objective function for each *inner layer* task learning as:

$$J^{PPO}_{\mathcal{T}_i}(\theta_i) = J^{C}_{ta_i}(\theta_i) - c_1 J^{VE}_{\mathcal{T}_i}(\theta_i), \tag{40}$$

where c_1 is the coefficient of value function loss. The outer layer objective is expressed as:

$$J^{MLD}(\theta) = \mathbb{E}_{\mathcal{T}_i \sim \rho(\mathcal{T}), \tau \sim P_{\mathcal{T}_i}(\tau, \theta'_i)} [J^{PPO}_{\mathcal{T}_i}(\theta'_i)], \tag{41}$$

where $\theta'_i = Up_{\tau \sim P_{\mathcal{T}_i}(\tau, \theta_i)}(\theta_i, \mathcal{T}_i), \theta_i = \theta$. We adopt the fist-order to approximate the second-order derivatives to save some calculation, which is defined as follows:

$$Grad^{MLD} := \frac{1}{n} \sum_{i=1}^{n} \Big[(\theta'_i - \theta)/\alpha/m \Big], \tag{42}$$

where we get n samples learning tasks in the *outer loop*, α is the learning rate of *inner loop* training, and m is the conducted gradient steps for the *inner loop* training.

Main Algorithm
Require: Workflow distribution: Λ, NN learning rates: $\alpha, \beta \sim \mathbb{R}^+$
Initial policy π_θ and initialization of $\mathcal{D} \leftarrow \varnothing$
Require: Environment sample's number: N
for $i = 1, \ldots N$ **do**
Use learned policies $\pi_{\theta'_H}$ to sample $\mathcal{D}' \sim \Lambda$
Add samples: $\mathcal{D} \leftarrow \mathcal{D}'$
for $j = 1, \ldots, H$ **do**
Set a sliding door with the size $h_i \in (0, H)$
Use policies π_θ to sample trajectories within first h_i samples $\tau_{h_i} \sim H$
Use τ_H with **PAC-RL** to calculate adapted parameters:
$\theta'_{h_i} = \theta + \alpha \sum_{j=1}^{h_i} \nabla \log \pi_\theta(s_t, a_t) r_{T_j}(s_t, a_t)$
Use adapted policy $\pi_{\theta'_{h_i}}$ sample trajectories $\tau'_{h_i} \sim H$
end for
Use τ_H to calculate adapted parameters:
$\theta'_H = \theta + \alpha \sum_{h_i} \nabla \log \pi_{\theta_{h_i}}(s_t, a_t) r_{T_j}(s_t, a_t)$
Use adapted policy θ'_H sample trajectories $\tau'_H \sim \mathcal{D}$
end for
Calculate update:
$\theta \leftarrow \theta - \beta \nabla_\theta \frac{1}{H} \sum_{j=1}^{H} \mathcal{L}_j(\theta'_H)$ using τ'_H
returnθ as θ'

Algorithm 1 Main Algorithm

```
for (s, a) do
  | Q(s, a) ← U(s, a), r(s, a) ← 0, n(s, a) ← 0
  | for s' ∈ S do
  | | | n(s, a, s') ← 0
  | end for
  | for 1, 2, 3, ... do
  | | if n(s, a) < m then
  | | | | n(s, a) ← n(s, a) + 1, r(s, a) ← r(s, a) + r, n(s, a, s') ← n(s, a, s') + 1
  | | | | if n(s, a) = m then
  | | | | | for i = 1, 2, 3, .., ln(1/(ε_1(1−Γ)))/(1−Γ) do
  | | | | | | for (s̄, ā) do
  | | | | | | | if (s̄, ā) > m then
  | | | | | | | | | Q(s̄, ā) ← R̂(s̄, ā) + ΓΣ_{s'} T̂(s'|s̄, ā) max_{a'} Q(s', a')
  | | | | | | | | | end if
  | | | | | | | end for
  | | | | | | end for
  | | | | end if
  | | end if
  | end for
end for
```

Algorithm 2 PAC-RL AlgorithmAlgorithm

This section describes the detailed process of the MLR-LC-DRLO algorithm, integrating and going through each part of the methodology formulated previously. As is shown in Algorithm 1, the input includes distribution over tasks, learning rates of the outer and inner loop. The meta policy neural network parameters are denoted as θ. We firstly sample a batch of learning tasks $\mathcal{T}$ with batch size n and conduct *inner loop* training for each sampled learning task. The *inner loop* training is conducted based on the PAC-Reinforcement-Learning we formulated aforementioned. The first step is the initialization of the algorithm: setting the initial parameters of the policy model and resetting the data set $\mathcal{D}$. Then is the sampling step: based on the number of environments, N data trajectories are sampled from the distribution Λ according to the current policy model and added to the data set $\mathcal{D}$. The following inner layer learning loop from are PAC-RL-based learning processes; sampling data sets τ_H inside $\mathcal{D}$ to calculate updated θ'_H based on each loss function with PAC-RL. When the PAC-RL converges or reaches the upper bound of the exploration, unlike conventional RL or other learning methods, the overall policy model is not updated by inner layer learning agent. After achieving updated θ'_H, RL agent uses θ'_H model to sample new data samples τ'_H from $\mathcal{D}$. After this, the algorithm turns to the outer learning layer, and the meta learner uses θ'_H to calculate loss function based on τ'_H to achieve an update of the overall policy model. In the next section we will evaluate MLR-LC-DRLO's performance.

Evaluation

Evaluation Measurements

We define the measurements as follows:

Offloading Latency-Critical Measurements We define several measurements to indicate and compare different experimental results and investigate different metrics specifically. One group is related to latency missing rate and offloading performance:

QoS-Latency-Critical Rate (QLCR) [6]: total percentage of executed tasks that meet latency required by QoS.

Expected-Latency-Critical Rate (ELCR) [6]: total percentage of executed tasks that meet expected latency. ELCR indicates the level of latency-critical for each method.

Necessary Training Iterations(NTI): the training iterations needed for convergence of policy model in an environment.

Offloading Robustness Measurements Robustness measurements includes: Dynamic Pressure Index(DPI), Offloading Performance Deviation(OPD) and Adaptation Steps and Data Usage for Performance Recovery(ASDUPR). They are formulated as follows: Dynamic Pressure Index(DPI): the indicator of the dynamic level of the current environment, including portion of workload change, latency change. It is defined as follows:

$$DPI = \frac{|WOR_{after} - WOR_{before}|}{WOR_{before}} \times 100\% \tag{43}$$

where WOR_{before}, WOR_{after} denotes the instant workload before and after the dynamics respectively. DPI shows the pressure level the system currently is having brougt by the dynamics. For OPD:

$$OPD = \frac{-(PER_{after} - PER_{before})}{PER_{before}} \times 100\% \tag{44}$$

where, PER_{after} denotes the instant average offloading latency after the influence of dynamic, PER_{before} indicates the previous converged average offloading latency value. Besides the instant performance deviation, ASDUPR is proposed to describe adaptation, includes time and data iteration needed for adaptation after performance deviation incurred by dynamics:

$$ASDUPR = OPD * ITER * t^o \tag{45}$$

where $ITER$ demonstrates the iteration time, t^o describes time spent for each iteration.

Based on the metrics defined previously, we implement comprehensive evaluation to validate robustness of MLR-LC-DRLO. Throughout the implementations do we aim to evaluate our proposed MLR-LC-DRLO in next section.

Set up

The configuration of the implementation consists of two parts: the configuration of the platform, shown in Table 2, and the configuration of simulation model, shown in Table 3.

Simulation Environment: We consider a cellular network, where the data transmission rate varies with the UE position. The CPU clock speed of UE, f_{UE} is set to 1GHz. There are four cores in each VM of the MEC host with a CPU clock speed of 2.5 GHz per core. The CPU clock speed of a VM, f_{VM} is $4 \times 2.5 = 10$ GHz. We implement a synthetic DAG generator according to [20] based on four parameters: *n*, *fat*, *density*, and *ccr*, where *n* represents the task number, *fat* controls the width and height of the DAG, *density* decides the number of edges between two levels of the DAG, and *ccr* denotes the ratio between the communication and computation cost of tasks (Table 4 and 5).

Results

As is shown in Table 6, we change same share of workload to show and compare the latency-critical offloading performance of our MLR-LC-DRLO algorithm against fine-tuned DQN, Double-DQN and CEM approaches on the same DAG data. More specifically, we add dynamic to scheduling by increasing workload for each method while keeping the the same resource availability setup. Then we assess the average latency rates of scheduled tasks to compare the performance robustness of the proposed MLR-LC-DRLO offloading against other methods. As is shown, we put the items in bold, which perform best in each row. Overall, compared with the fine-tuned DQN, Double-DQN and CEM methods, our algorithm MLR-LC-DRLO offers more stable latency-critical offloading performance every time after dynamic influence in the environment. More specifically, MLR-LC-DRLO outperforms the fine-tuned DQN, Double-DQN and CEM approaches in the latency rates and necessary training iteration. From the perspective of latency rate, averagely more than $95.33\% \pm 0.34\%$ tasks offloaded by MLR-LC-DRLO finish their execution with the shorter latency than the expected latency 720ms [20]. While tasks offloaded by other RL-based methods finish their execution averagely with a range of $8.5 - 20\%$ violation rate of latency requirement. Moreover, when given heavier workflows (topology 2, n=30, UT=DT=5.5Mbps), shown in Table 6, our method MLR-LC-DRLO still offers workflows more stable latency-critical offloading performance, which is more percentage of tasks finish execution with the lower latency than the expected one under dynamics from the environments. After all MLR-LC-DRLO outperforms the fine-tuned DQN, Double-DQN and CEM in performance stability.

As shown in Figs. 5 and 6, we show the change of OPD and ASDUPR under different dynamics to show the robustness of MLR-LC-DRLO from the perspective of offloading performance stability and the expense taken for the recovery from the performance deviation. Firstly, from the perspective of offloading performance stability, as shown in Fig. 5: we increase the workload with same percentage for all the offloading approaches. The performance deviation of our proposed MLR-LC-DRLO always remains stable within 55% throughout different workload environments, in some environment the deviation is even under 25%. In contrast, fine-tuned DQN, Double-DQN and CEM approaches' performance deviation range is rather broader between 50% and even beyond 300% with the same portion of increased workload. Therefore, the offloading performance stability of our MLR-LC-DRLO outperforms the conventional RL-based offloading approaches. Then from the perspective of performance deviation recovery, as shown in Fig. 6 where we compare the adaptation speed

Table 2 Platform settings: The cluster, where we implement the experiments, consists of 18 nodes, each node's configuration is shown in the table. The software environment includes: Anaconda, python-numpy, python-scipy, python-dev, python-pip, python-nose, g++ libopenblas-dev, git, Thensorflow, and python-matplotlib

Components	GTX 1080 Ti	Intel(R) Xeon(R) Gold 5118	CPU	Memory	Local HDD	Local SSD
Configuration						
	4 x	2 x	@ 2.30GHz (12 cores per cpu)	128 GB	2 x 10 TB	2 x 4 TB

Table 3 Simulation set up: We generate synthetic DAG according to [49], whose model is characterised by: *n*, *fat*, *density*, and *ccr*, where *n* represents the task number, *fat* controls the width and height of the DAG, *density* decides the number of edges between two levels of the DAG, and *ccr* denotes the ratio between the communication and computation cost of tasks

Items	f_{UE}	Per VM	DAG-task number	DAG-width and height	DAG-density	DAG-task cost
Configuration						
	1 GHz	4 x 2.5GHz	*n*	*fat*	*density*	*ccr*

Table 4 Fine-tuned baseline approaches: we train DQN, Double-DQN, CEM based approaches as baselines of our proposed MLR-LC-DRLO

Fine-tuned RL Approaches						
Parameters	**NN Layers**	**Replay Buffer Size**	**Optimizer**	ρ	**Learning Rate**	**Activation Function**
Baseline Approaches						
DQN	4	–	*Adam*	0.95	1e-3	ReLU, Softmax
Double-DQN	3	500	*Adam*	0.95	1e-3	ReLU, Softmax
CEM	3	–	*Adam*	0.95	1e-3	ReLU, Softmax

Table 5 MLR-LC-DRLO Hyperparameter set up

Hyperparameter	**Set up**	**Hyperparameter**	**Set up**
Encoder NN	LSTM, 2 Layers, norm: on	Outer Learning Rate	5×10^{-4}
NN Neuron Amount	256	Activation Function	tanh
Decoder NN	LSTM, 2 layers, norm: on	Loss Coefficient	0.5
Inner Learning Rate	5×10^{-4}	Slice Constant	0.2
Optimizer	Adam	Discount Factor	0.99
Gradient step m	3	Adv Discount Factor	0.95

or performance recovery speed discounted by the performance deviation portion, which balances the adaptation speed and robustness performance. As shown, we cloud see that the adaptation speed of MLR-LC-DRLO is more than five times faster than fine-tuned DQN, Double-DQN and CEM averagely after every time increase of workload, at some point, even more, proving its robustness to dynamics of the environment.

Discussion

As shown in the result section, compared with conventional RL-based approaches, our proposed offloading approach MLR-LC-DRLO shows advantages in terms of the offloading performance robustness and recovery speed after influence from dynamics among heterogeneous environments. More specifically, the offloading performance deviation and adaptation speed of our proposed approach MLR-LC-DRLO show a stable pattern of change during increased workload. When the dynamic change of the DPI is within 30%-50%, both offloading performance deviation and adaptation speed increase with DPI; When the dynamic change of the DPI is in a range of 30%-50%, both offloading performance deviation and adaptation speed decrease with DPI; When the dynamic change of the DPI is beyond 50%, both offloading performance deviation and adaptation speed increase again. Overall, when the DPI is within 50%, MLR-LC-DRLO could stay robustness with lower than 30% performance deviation. When DPI goes beyond 50%, the performance deviation of MLR-LC-DRLO still stay within 50%. The robustness starts to decrease when DPI beyond 50% but still with lower than 50% performance deviation, much lower than fine-tuned RL methods (more than 300%). One of our future work directions is to expand the robustness range against the dynamics, that is keeping lower performance deviation against wider range of DPI change. Another direction of future work is to reduce the instant offloading performance deviation right after the DPI changes. Furthermore, by investigating exploration strategies of the RL framework, we could better control the training time and accuracy. We are currently investigating the exploration and exploitation accuracy of RL-based approaches.

Regarding the superiority achieved by our proposed methods, there are two main aspects of the insight, the first one is the merit that Meta-learning can leverage prior knowledge from previous tasks to improve learning on new tasks. In this way, the prior offloading knowledge can be accumulated and transferred to the following phase. By analyzing patterns and relationships across multiple environments, a meta-learning model can identify commonalities and transfer knowledge from one environment to another. This transfer learning can help a model learn new offload patterns more efficiently and effectively. Also, meta-learning can help avoid overfitting to specific training data by learning a more generalizable learning strategy. By training on multiple tasks, the meta-learning model can learn to generalize across tasks and avoid overfitting to specific examples. This can lead to a more adaptive model that can perform well on a wide range of tasks and data.

The other aspect is Probably Approximate Correct (PAC), which is a framework in machine learning that aims to balance the accuracy of a model with the amount of data needed to achieve that accuracy. The PAC framework provides a way to measure the sample complexity

Table 6 Offloading Performance Comparison: we compare MLR-LC-DRLO with fine-tuned DQN, Double-DQN and CEM to show that MLR-LC-DRLO achieves better latency-critical offloading performance

Approaches	**MLR-LC-DRLO**			**DQN**			**Double-DQN**			**CEM**		
Indicators	**QLCR**	**ELCR**	**NTI**	**QLCR**	**ELCR**	**NTI**	**QLCR**	**ELCR**	**NTI**	**QLCR**	**ELCR**	**NTI**
Workflow Topology												
Topology 1	**98.33%±0.56%**	**96.13%±3.24%**	**537±32**	95.33%±1.78%	90.22±2.23%	**653 ± 28**	96.12%±0.65%	93.23±2.15%	611 ± 34	87.55% ± 2.76%	85.31% ± 5.82%	826 ± 57
Topology 2	**97.01%±1.28%**	**95.66%±2.46%**	**581±17**	93.92%±1.78%	88.67±3.34%	**692 ± 36**	95.55%±1.38%	92.14±3.24%	649 ± 34	83.84% ± 3.57%	80.06% ± 1.36%	864 ± 39
n=20	**97.33%±1.14%**	**95.88%±2.16%**	**553±26**	94.52%±2.63%	89.53±3.25%	**633 ± 37**	95.36%±0.91%	91.53±2.26%	603 ± 11	89.98% ± 1.36%	87.43% ± 3.35%	638 ± 42
n=30	**95.33%±0.34%**	**94.27%±1.24%**	**569±31**	91.57%±3.21%	88.56±1.53%	**687 ± 45**	93.66%±1.80%	91.08±1.30%	579 ± 43	86.50% ± 1.09%	85.66% ± 2.57%	712 ± 35
UT=DT=8.5Mbps	**98.12%±0.68%**	**96.77%±2.15%**	**493±46**	93.58%±2.20%	91.65±1.10%	586 ± 35	97.39%±0.60%	95.42±3.63%	523 ± 52	89.33% ± 1.55%	86.45% ± 2.33%	721 ± 22
UT=DT=5.5Mbps	**96.06%±1.06%**	**95.69%±1.53%**	**556±32**	91.06%±2.73%	90.32±0.34%	**635 ± 30**	95.53%±1.66%	93.65±1.98%	609 ± 25	85.34% ± 3.87%	82.68% ± 2.08%	865 ± 48

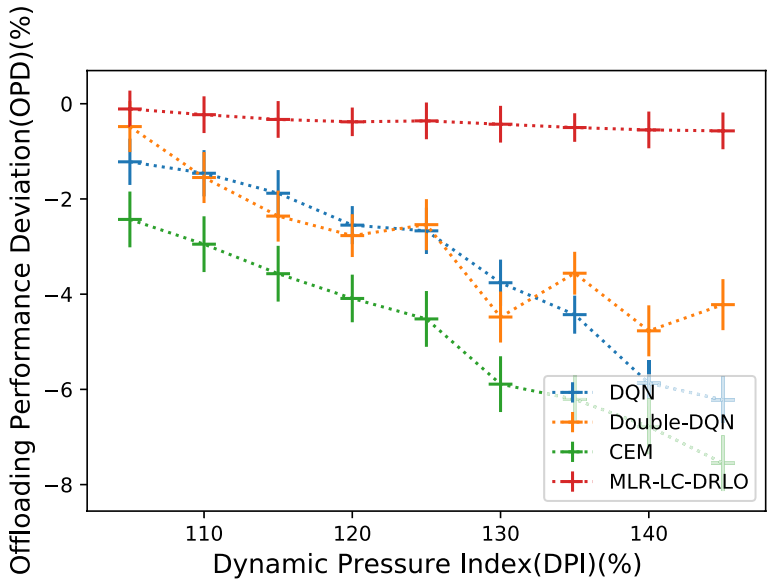

Fig. 5 Performance deviation of MLR-LC-DRLO and other 3 fine-tuned other fine-tuned RL-based approaches: with the same amount of DPI, MLR-LC-DRLO experiences lower performance deviation

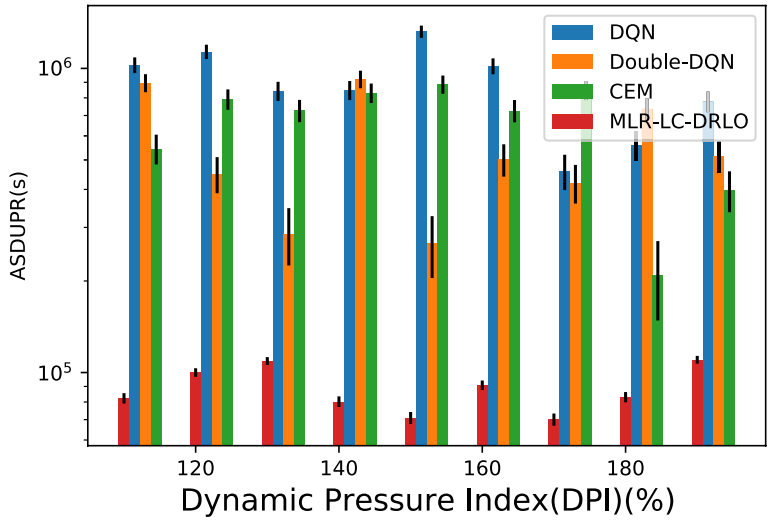

Fig. 6 Adaptation time spent on retraining after workload changes between MLR-LC-DRLO and other fine-tuned RL-based approaches: with the same amount of DPI, MLR-LC-DRLO spend less time to recover offloading performance

of a learning algorithm, which is the number of training examples needed to achieve a certain level of accuracy. One advantage of the PAC framework is that it can lead to faster convergence of learning algorithms. The PAC framework is designed to ensure that a learning algorithm will be able to generalize well from the training data to new, unseen data. To achieve this, the PAC framework requires that the algorithm be able to achieve a certain level of accuracy with high probability, meaning that the algorithm should be able to correctly classify most of the test examples with high confidence. This requirement ensures that the algorithm will perform well on new data, even if it has not seen those examples during training.

In addition, the PAC framework provides a way to measure the sample complexity of a learning algorithm. This measure is based on the required level of accuracy and the confidence level, and provides a way to estimate the number of training examples needed to achieve the desired level of accuracy. This allows researchers to compare different learning algorithms and choose the

one with the lowest sample complexity, which can lead to faster convergence and more efficient learning.

As of the limitation of our work, one is the real-time adaptation efficiency. Currently, the algorithm is trained offline then adapt to a new environment. We plan to integrate an online-offline switch scheme in the future to improve the real-time adaptation efficiency. Also, more implementations of real world data is also part of our future steps.

Conclusion

In this work, MLR-LC-DRLO, a robust task scheduling framework, is presented to offer latency-guaranteed scheduling for time-critical tasks and improve the schedule's robustness in the meantime. We propose a meta-gradient robust reinforcement learning framework to quickly adapt a scheduling policy model to a newly changed environment while using a PAC-based latency-critical RL scheme to maintain the latency guarantee. Experimental results show that our approach can provide the latency guarantee, outperforming fine-tuned RL methods. Furthermore, our MLR-LC-DRLO approach finishes adaptation in new environments using fewer training iterations, 2× to 5× faster than the fine-tuned RL approach, achieving better robustness while offering latency guarantees.

Authors' contributions

Hongyun Liu: Conceptualization, Methodology, Software, Writing- Original draft preparation. Ruyue Xin: Conceptualization, Methodology, Software, Writing- Original draft preparation. Peng Chen: Conceptualization, Methodology, Software, Writing, Data curation, Writing- Original draft preparation.Hui Gao: Methodology, Software, Writing, Data curation, Writing. Paola Grosso: Supervision, Writing- Reviewing and Editing. Zhiming Zhao: Supervision, Writing- Reviewing and Editing. All authors read and approved the final manuscript.

Funding

This work is funded by the European Union's Horizon 2020 projects: ARTICONF (Grant No. 825134), ENVRI-FAIR project (Grant No. 824068), BLUECLOUD (Grant No. 862409), Bluecloud2026(Grant No. 101094227) and LifeWatch ERIC, the Natural Science Foundation of Shaanxi (Grant No. 2022JQ-651), China Scholarship Council, Science and Technology Program of Sichuan Province (Grant No.2020YFG0326), and Talent Program of Xihua University (Grant No.Z202047).

Availability of data and materials

Not applicable.

Declarations

Ethics approval and consent to participate

Not applicable.

Competing interests

The authors declare no competing interests.

Received: 4 November 2022 Accepted: 2 April 2023
Published online: 15 April 2023

References

1. Wu L, Liu M, Wang XM, Chen Gh, Hg Gong (2011) Mobile distribution-aware data dissemination for vehicular ad hoc networks. Ruanjian Xuebao/J Softw 22(7):1580–1596
2. Pham QV, Fang F, Ha VN, Piran MJ, Le M, Le LB, Hwang WJ, Ding Z (2020) A survey of multi-access edge computing in 5g and beyond: Fundamentals, technology integration, and state-of-the-art. IEEE Access 8:116974–117017
3. Song C, Liu M, Cao J, Zheng Y, Gong H, Chen G (2009) Maximizing network lifetime based on transmission range adjustment in wireless sensor networks. Comput Commun 32(11):1316–1325
4. Yu S, Wang X, Langar R (2017) Computation offloading for mobile edge computing: A deep learning approach. In: 2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC). IEEE, New York, pp 1–6
5. Wang J, Hu J, Min G, Zhan W, Ni Q, Georgalas N (2019) Computation offloading in multi-access edge computing using a deep sequential model based on reinforcement learning. IEEE Commun Mag 57(5):64–69
6. Liu H, Chen P, Zhao Z (2021) Towards a robust meta-reinforcement learning-based scheduling framework for time critical tasks in cloud environments. In: 2021 IEEE 14th International Conference on Cloud Computing (CLOUD). IEEE, pp 637–647
7. Liu H, Xin R, Chen P, Zhao Z (2022) Multi-objective robust workflow offloading in edge-to-cloud continuum. In: 2022 IEEE 15th International Conference on Cloud Computing (CLOUD). IEEE, pp 469–478
8. Singh S, Dhillon HS, Andrews JG (2013) Offloading in heterogeneous networks: Modeling, analysis, and design insights. IEEE Trans Wirel Commun 12(5):2484–2497
9. Zhang K, Mao Y, Leng S, Zhao Q, Li L, Peng X, Pan L, Maharjan S, Zhang Y (2016) Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks. IEEE Access 4:5896–5907
10. Chen C, Li H, Li H, Fu R, Liu Y, Wan S (2022) Efficiency and fairness oriented dynamic task offloading in internet of vehicles. IEEE Trans Green Commun Netw 6(3):1481–1493. https://doi.org/10.1109/TGCN.2022.3167643
11. Chen C, Zeng Y, Li H, Liu Y, Wan S (2023) A multihop task offloading decision model in mec-enabled internet of vehicles. IEEE Internet Things J 10(4):3215–3230. https://doi.org/10.1109/JIOT.2022.3143529
12. Wei W, Yang R, Gu H, Zhao W, Chen C, Wan S (2022) Multi-objective optimization for resource allocation in vehicular cloud computing networks. IEEE Trans Intell Transp Syst 23(12):25536–25545. https://doi.org/10.1109/TITS.2021.3091321
13. Ye Y, Hu RQ, Lu G, Shi L (2020) Enhance latency-constrained computation in mec networks using uplink noma. IEEE Trans Commun 68(4):2409–2425
14. Feng J, Pei Q, Yu FR, Chu X, Shang B (2019) Computation offloading and resource allocation for wireless powered mobile edge computing with latency constraint. IEEE Wirel Commun Lett 8(5):1320–1323
15. Meng H, Chao D, Guo Q (2019) Deep reinforcement learning based task offloading algorithm for mobile-edge computing systems. In: Proceedings of the 2019 4th International Conference on Mathematics and Artificial Intelligence. Association for Computing Machinery, New York, pp 90–94
16. Min M, Xiao L, Chen Y, Cheng P, Wu D, Zhuang W (2019) Learning-based computation offloading for iot devices with energy harvesting. IEEE Trans Veh Technol 68(2):1930–1941
17. Dinh TQ, La QD, Quek TQ, Shin H (2018) Learning for computation offloading in mobile edge computing. IEEE Trans Commun 66(12):6353–6367
18. Cheng N, Lyu F, Quan W, Zhou C, He H, Shi W, Shen X (2019) Space/aerial-assisted computing offloading for iot applications: A learning-based approach. IEEE J Sel Areas Commun 37(5):1117–1129
19. Li M, Yu FR, Si P, Wu W, Zhang Y (2020) Resource optimization for delay-tolerant data in blockchain-enabled iot with edge computing: A deep reinforcement learning approach. IEEE Internet Things J 7(10):9399–9412
20. Wang J, Hu J, Min G, Zomaya AY, Georgalas N (2020) Fast adaptive task offloading in edge computing based on meta reinforcement learning. IEEE Trans Parallel Distrib Syst 32(1):242–253
21. Cao Z, Zhou P, Li R, Huang S, Wu D (2020) Multiagent deep reinforcement learning for joint multichannel access and task offloading of mobile-edge computing in industry 4.0. IEEE Internet Things J 7(7):6201–6213

22. Lu H, Gu C, Luo F, Ding W, Liu X (2020) Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning. Futur Gener Comput Syst 102:847–861
23. Lolos K, Konstantinou I, Kantere V, Koziris N (2017) Elastic management of cloud applications using adaptive reinforcement learning. In: 2017 IEEE International Conference on Big Data (Big Data). IEEE, pp 203–212
24. Guan S, Boukerche A, Loureiro A (2020) Novel sustainable and heterogeneous offloading management techniques in proactive cloudlets. IEEE Trans Sustain Comput 6(2):334–346
25. Li W, Jin S (2021) Performance evaluation and optimization of a task offloading strategy on the mobile edge computing with edge heterogeneity. J Supercomput 77(11):12486–12507
26. Xiong K, Leng S, Huang C, Yuen C, Guan YL (2020) Intelligent task offloading for heterogeneous v2x communications. IEEE Trans Intell Transp Syst 22(4):2226–2238
27. Mach P, Becvar Z (2017) Mobile edge computing: A survey on architecture and computation offloading. IEEE Commun Surv Tutor 19(3):1628–1656
28. Zhao Z, Zhao R, Xia J, Lei X, Li D, Yuen C, Fan L (2019) A novel framework of three-hierarchical offloading optimization for mec in industrial iot networks. IEEE Trans Ind Inform 16(8):5424–5434
29. Huang M, Liu W, Wang T, Liu A, Zhang S (2019) A cloud-mec collaborative task offloading scheme with service orchestration. IEEE Internet Things J 7(7):5792–5805
30. Yang X, Yu X, Huang H, Zhu H (2019) Energy efficiency based joint computation offloading and resource allocation in multi-access mec systems. IEEE Access 7:117054–117062
31. Chen X, Zhang H, Wu C, Mao S, Ji Y, Bennis M (2018) Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning. IEEE Internet Things J 6(3):4005–4018
32. Chen M, Guo S, Liu K, Liao X, Xiao B (2020) Robust computation offloading and resource scheduling in cloudlet-based mobile cloud computing. IEEE Trans Mob Comput 20(5):2025–2040
33. Hyytiä E, Spyropoulos T, Ott J (2015) Offload (only) the right jobs: Robust offloading using the markov decision processes. In: 2015 IEEE 16th international symposium on a world of wireless, mobile and multimedia networks (WoWMoM). IEEE, pp 1–9
34. Fiechter CN (1994) Efficient reinforcement learning. In: Proceedings of the seventh annual conference on Computational learning theory. pp 88–97
35. Dann C, Li L, Wei W, Brunskill E (2019) Policy certificates: Towards accountable reinforcement learning. In: International Conference on Machine Learning. PMLR, pp 1507–1516
36. Ménard P, Domingues OD, Jonsson A, Kaufmann E, Leurent E, Valko M (2021) Fast active learning for pure exploration in reinforcement learning. In: International Conference on Machine Learning. PMLR, pp 7599–7608
37. Domingues OD, Ménard P, Kaufmann E, Valko M (2021) Episodic reinforcement learning in finite mdps: Minimax lower bounds revisited. In: Algorithmic Learning Theory. PMLR, pp 578–598
38. Azar MG, Osband I, Munos R (2017) Minimax regret bounds for reinforcement learning. In: International Conference on Machine Learning. PMLR, pp 263–272
39. Simchowitz M, Jamieson KG (2019) Non-asymptotic gap-dependent regret bounds for tabular mdps. Adv Neural Inf Process Syst 32:1153–1162
40. Xu H, Ma T, Du S (2021) Fine-grained gap-dependent bounds for tabular mdps via adaptive multi-step bootstrap. In: Conference on Learning Theory. PMLR, pp 4438–4472
41. Dann C, Marinov TV, Mohri M, Zimmert J (2021) Beyond value-function gaps: Improved instance-dependent regret bounds for episodic reinforcement learning. Adv Neural Inf Process Syst 34:1–12
42. Wagenmaker AJ, Simchowitz M, Jamieson K (2022) Beyond no regret: Instance-dependent pac reinforcement learning. In: Conference on Learning Theory. PMLR, pp 358–418
43. Zaremba W, Sutskever I, Vinyals O (2014) Recurrent neural network regularization. arXiv preprint arXiv:1409.2329
44. Bahdanau D, Cho K, Bengio Y (2014) Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473
45. Sutskever I, Vinyals O, Le QV (2014) Sequence to sequence learning with neural networks. Adv Neural Inf Process Syst 27
46. Finn C, Abbeel P, Levine S (2017) Model-agnostic meta-learning for fast adaptation of deep networks. In: International conference on machine learning. PMLR, pp 1126–1135
47. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347
48. Schulman J, Moritz P, Levine S, Jordan M, Abbeel P (2015) High-dimensional continuous control using generalized advantage estimation. arXiv preprint arXiv:1506.02438
49. Arabnejad H, Barbosa JG (2013) List scheduling algorithm for heterogeneous systems by an optimistic cost table. IEEE Trans Parallel Distrib Syst 25(3):682–694

Publisher's Note