Decentralized Task Offloading in Edge Computing: An Offline-to-Online Reinforcement Learning Approach

Hongcai Lin, Lei Yang, Hao Guo, and Jiannong Cao, *Fellow, IEEE*

***Abstract*—Decentralized task offloading among cooperative edge nodes has been a promising solution to enhance resource utilization and improve users' Quality of Experience (QoE) in edge computing. However, current decentralized methods, such as heuristics and game theory-based methods, either optimize greedily or depend on rigid assumptions, failing to adapt to the dynamic edge environment. Existing DRL-based approaches train the model in a simulation and then apply it in practical systems. These methods may perform poorly because of the divergence between the practical system and the simulated environment. Other methods that train and deploy the model directly in real-world systems face a cold-start problem, which will reduce the users' QoE before the model converges. This paper proposes a novel offline-to-online DRL called (O2O-DRL). It uses the heuristic task logs to warm-start the DRL model offline. However, offline and online data have different distributions, so using offline methods for online fine-tuning will ruin the policy learned offline. To avoid this problem, we use on-policy DRL to fine-tune the model and prevent value overestimation. We evaluate O2O-DRL with other approaches in a simulation and a Kubernetes-based testbed. The performance results show that O2O-DRL outperforms other methods and solves the cold-start problem.**

***Index Terms*—Edge computing, decentralized task offloading, deep reinforcement learning (DRL).**

I. Introduction

WITH the proliferation of the Internet of Things devices (e.g., wearable devices, smartphones), many new computation-intensive and latency-sensitive applications (e.g., face recognition, augmented reality) have recently emerged. Due to the demand for high network bandwidth in these applications, edge computing (EC) becomes a preferable paradigm for processing data near end-users at the network edge. However, edge nodes are typically limited in computation resources. Yet, the computation workload in edge nodes is highly dynamic and geographically imbalanced. Therefore, it is challenging for an individual edge node to provide satisfactory computation service all the time. Fortunately, we can exploit cooperation among edge nodes [1] to enhance resource utilization and improve users' Quality of Experience (QoE). For example, overloaded edge nodes can forward part of their workload to other edge nodes with light computation, thereby balancing workload among geographically distributed edge nodes.

Most of the literature on task offloading attempts to minimize the latency or energy consumption [2], [3], [4], [5] in collaborative edge computing. However, each operational unit in network systems, such as edge nodes and transmission links, is unstable and may fail while processing tasks. Therefore, the task offloading scheduler in edge nodes should be more intelligent to balance the latency and reliability of tasks.

Existing solutions in collaborative edge computing can be divided into two categories, centralized approaches, and decentralized approaches. Note that these methods only optimize latency or energy consumption without considering reliability. Specifically, the centralized approaches [6], [7], [8] require global knowledge of the system state to make the task offloading decision of each task in the edge network. Nevertheless, up-to-date global knowledge of fast-changing information, such as task information and the workload of edge nodes, would require prohibitive transmission overhead, which is unrealistic in practical large-scale networks. Traditional decentralized heuristic approaches [9], [10] are based on greedy optimization and may trap into suboptimal from a long-term perspective. On the other hand, game theory-based methods [4] heavily rely on rigid assumptions to calculate the static Nash Equilibrium (NE), making it difficult to characterize real-world scenarios comprehensively.

Recently, Deep Reinforcement Learning (DRL) has attracted significant attention in both academia and industry [11]. DRL-based task offloading approaches can enjoy the following advantages. First, DRL can adaptively make task-offloading decisions based on the dynamic environment. Second, DRL aims to maximize the discounted cumulated reward to optimize

Manuscript received 23 February 2023; revised 4 February 2024; accepted 9 March 2024. Date of publication 19 March 2024; date of current version 10 May 2024. This work was supported in part by the Guangdong Basic and Applied Basic Research Foundation under Grant 2022A1515010374; in part by the Hong Kong RGC Theme-based Research Scheme (TRS) under Grant T43-513/23-N; in part by HK RGC Collaborative Research Fund under Grant C5018-20G; and in part by HK RGC General Research Fund under Grant PolyU-15204921 and PolyU-15220922. Recommended for acceptance by H. Jiang. *(Corresponding author: Lei Yang.)*

Hongcai Lin, Lei Yang, and Hao Guo are with the School of Software Engineering, South China University of Technology, Guangzhou 510006, China (e-mail: selhc@mail.scut.edu.cn; sely@scut.edu.cn; seguohao@mail.scut.edu.cn).

Jiannong Cao is with the Department of Computing, Hong Kong Polytechnic University, Hong Kong, China (e-mail: jiannong.cao@polyu.edu.hk).

Digital Object Identifier 10.1109/TC.2024.3377912

0018-9340 © 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Hong Kong Polytechnic University. Downloaded on May 27,2024 at 10:28:24 UTC from IEEE Xplore. Restrictions apply.

the objective from a long-term perspective. However, existing DRL-based approaches [12] train the DRL model in the simulation environment and deploy the model in real-world systems. Since it is difficult to characterize the real-world system in the simulation environment, the model that achieves good results in the simulation environment may not be effective when deployed to the actual system.

Alternative methods [1], [2], [3], [13] directly train and deploy the DRL model in the edge computing environment, which is also called online learning. Online learning usually encounters the cold-start problem, which might cause poor QoE to users before model convergence. The recent offline DRL [14], [15] provides another practical way to apply DRL to edge computing. Offline DRL trains the DRL model using previously collected datasets without further interaction with the environment. Therefore, the offline DRL model can be deployed in the edge computing environment for task offloading. However, the behavior policy of datasets may be sub-optimal, and we cannot correct the learning policy via interaction with the environment, leading to an unsatisfactory performance offline DRL model. Therefore, it is necessary to fine-tune the DRL when deploying the DRL model in a real-world system.

In this paper, we study the problem of decentralized task offloading in edge computing. Each edge node can process the arrival tasks locally or offload them to other edge nodes based on local information. First, we formalize this problem as a Decentralized Partially Observable Markov Decision Process (Dec-POMDP). Then, we propose an offline-to-online DRL solution called O2O-DRL. O2O-DRL first trains a DRL model in the offline phase using heuristic task running logs to warm-start the DRL model. Then, in the online stage, due to the distribution shift between offline and online data [14], [16], directly applying offline DRL in online learning will lead to the high bias estimation of unseen observations and destroy the good policy obtained via offline DRL. Therefore, O2O-DRL uses on-policy DRL to fine-tune the model and avoid performance degradation in the initial online phase.

In sum, our main contributions can be summarized as follows:

- We consider the problem of decentralized task offloading among cooperative edge nodes. Each edge node makes the task offloading decision independently based on local information. We are the first to consider the reliability of tasks during execution and transmission in this decentralized edge network.
- We formalize the decentralized task offloading problem as a Dec-POMDP and propose a novel offline-to-online DRL solution called Offline-to-Online DRL (O2O-DRL). Our approach can build upon most prior heuristic algorithms to enhance their performance. To our knowledge, this is the first work to solve the cold-start problem when applying DRL in edge computing.
- We compare O2O-DRL with benchmark approaches in a numerical simulation and a Kubernetes-based testbed. The performance results show that O2O-DRL has advantages in the success rate of tasks over state-of-the-art DRL methods and avoids the cold-start problem.

The rest of this paper is organized as follows. The related work is reviewed in Section II. The system model and problem formulation are presented in Section III. The details of the DRL approach are described in Section IV. Simulation results are presented in Section V. The experiment in the Kubernetes-based testbed is presented in Section VI. Finally, Section VII concludes the paper.

II. Related Work

Recent research [7], [17], [18], [19] has considered the cooperation between edge nodes. The authors in [17] study the joint multi-task partial computation offloading and network flow scheduling problem. They propose a heuristic algorithm to minimize the average completion time of all tasks. [18] addresses the peer offloading problem in mobile-edge computing, focusing on the optimization of time-average throughput under energy consumption and worst-case response time constraints. They propose two online algorithms to manage the stochastic arrival of computation tasks, with a particular emphasis on the often-neglected worst-case response time, a critical quality of service metric in real-time applications. However, these researches need centralized control, which may cause prohibitive transmission overhead, and are not scalable to large-scale edge networks.

Some work investigated the decentralized task offloading problem based on heuristic algorithms [9], [10], [20], [21]. In [9], the authors studied the task offloading problem in a sensor network, where each device aimed to minimize the execution latency and energy consumption. They proposed a reactive distributed algorithm to solve the problem. The author in [21] introduced an efficient load-balancing protocol for fog computing, which utilized a threshold-based mechanism to efficiently distribute computational tasks, adapting to node heterogeneity and reducing delay and overhead. However, those methods mainly focus on greedy optimization, which may be suboptimal solutions from a long-term perspective.

Some papers propose decentralized task offloading algorithms based on game theory [4], [5], [22]. For example, the authors in [4] investigated the task offloading problem in fog computing. They formulated the problem as multi-user computation offloading game and proposed a Static Mixed Nash Equilibrium (SM-NE) algorithm to derive the NE. Furthermore, [22] extended the scope of this research by employing a multileader multifollower Stackelberg game to delineate optimal pricing strategies for MEC servers and user data offloading strategies. Their approach uniquely incorporates user risk awareness, which is articulated through prospect-theoretic utility functions, and addresses the potential for server overexploitation by utilizing the theory of the commons. However, these approaches are built on rigid assumptions and may not characterize real-world scenarios adequately. Moreover, game theory-based methods must frequently recalculate the optimal policies, which would cause a non-negligible computational overhead at the execution phase.

Deep reinforcement learning (DRL) has recently attracted significant attention from academia to industry [1], [2], [11],

Authorized licensed use limited to: Hong Kong Polytechnic University. Downloaded on May 27,2024 at 10:28:24 UTC from IEEE Xplore. Restrictions apply.

TABLE I
MAIN NOTATIONS

Notations	Description
$\mathcal{V}$	The set of edge nodes in the network.
$\mathcal{E}$	The set of transmission links in the network.
N	The total number of edge nodes.
i	The index of an edge node.
e_{ij}	The transmission link between edge node i and j.
t	The index of time slot.
T	The total time slots.
λ_i	The task arrival probability of edge node i.
$K_i(t)$	The task generated at edge node i in time slot t.
$s_i(t)$	The input size of task $K_i(t)$.
$c_i(t)$	The CPU cycles of task $K_i(t)$.
$d_i(t)$	The deadline of task $K_i(t)$.
r_{ij}	The transmission rate from edge node i to j.
F_i	The computation capacity of edge node i.
α_i	The failure rate of edge node i.
β_{ij}	The failure rate of the transmission link e_{ij}.
o_t^i	The observation of edge node i in time slot t.
a_t^i	The action of edge node i in time slot t.
r_t	The team reward in time slot t.

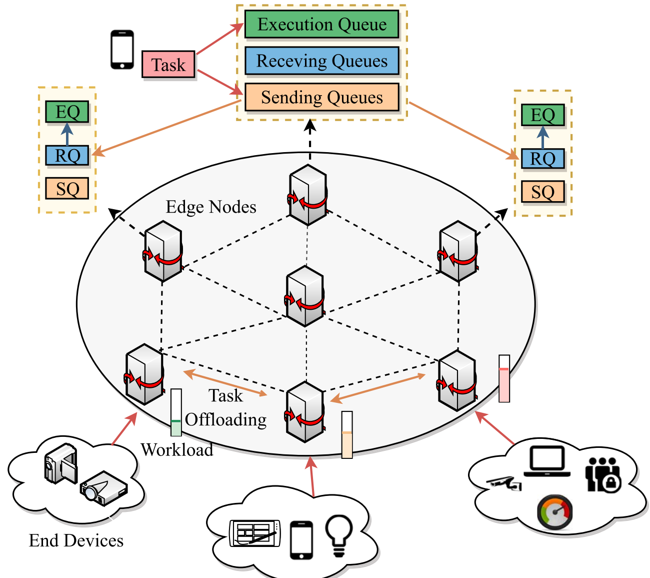


Fig. 1. **Overview of the system model.** The system includes a set of edge nodes that receive tasks from end devices. Each edge node maintains an execution queue to process tasks, receiving queues for receiving tasks from other edge nodes and sending queues for transmitting tasks to other edge nodes.

[23]. The authors in [2] studied the decentralized task offloading problem in a pervasive edge computing network that consists of multiple edge devices. A decentralized task offloading algorithm is proposed based on multi-agent imitation learning. The authors in [23] studied the joint task partitioning and power control problem in a fog computing network with multiple mobile devices (MDs) and fog devices (FDs). They proposed a multi-agent deep deterministic policy gradient (MADDPG) based task offloading algorithm to reduce the system utility. However, the above work directly applies DRL to edge computing, which will encounter the cold-start problem that may significantly affect the user's QoE before the DRL model converges. Offline DRL [14], [15] considers training the DRL model without interacting with the environment. The authors in [16] consider the offline-to-online problem by balancing the replay buffer between the offline and online buffer. However, it still suffers from a performance decline in part of their experiment.

In this paper, we consider decentralized task offloading among edge nodes. Different from prior works, we further consider the reliability of task execution and transmission reliability. Furthermore, to our knowledge, we are the first to solve the cold-start problem effectively in the context of edge computing with DRL.

III. System Model and Problem Definition

A. System Model

The system model is shown in Fig. 1. We consider a heterogeneous edge network that is modeled as a connected graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of edge nodes, $\mathcal{V} = \{i|1 \leq i \leq N\}$, and $\mathcal{E}$ is the set of links connecting different edge nodes, $\mathcal{E} = \{e_{ij}|i, j \in \mathcal{V}\}$. Note that this model can be applied to both wired and wireless scenarios. The time is slotted with a constant slot duration, and the time slot index can be denoted by t, where $t \in \{1, 2, ..., T\}$. Each edge node $i \in \mathcal{V}$ may receive computation-intensive and delay-sensitive tasks from end-users in each time slot. We assume that a new task arriving at edge node i within time slot t follows a Bernoulli distribution [24], [25], [26] with probability λ_i. $K_i(t)$ denotes the task arriving at the edge node i in time slot t, which can be described by a tuple $\{s_i(t), c_i(t), d_i(t)\}$. $s_i(t)$ is the size of the program code and input data; $c_i(t) = \delta_i(t) * s_i(t)$ [4], [27] is the number of CPU cycles required to finish the task, where $\delta_i(t)$ is the computation intensity that indicates the required CPU cycles per bit; $d_i(t)$ is the deadline of the task. Each edge node i can either process the arriving task locally or offload it to other edge nodes. Each edge node i maintains an execution queue with tasks scheduled based on First-In, First-Out (FIFO) order. Current tasks must wait in the execution queue if other tasks occupy the computation resources. We denote by F_i the computation capacity of edge node i. Thus, the computation time for the task $K_i(t)$ processed by the edge node j can be calculated as

$$T_{ij}^c(t) = \frac{c_i(t)}{F_j}, \tag{1}$$

Let r_{ij} denote the transmission rate between edge node i and j. Thus, the transmission time to send task $K_i(t)$ from edge node i to j can be computed by

$$T_{ij}^t(t) = \frac{s_i(t)}{r_{ij}}. \tag{2}$$

Each edge node maintains $N-1$ receiving queues for receiving tasks from other edge nodes and $N-1$ sending queues for transmitting tasks to other edge nodes. The tasks are transferred following the FIFO order. After task $K_i(t)$ is received by edge node j, it waits in the execution queue for processing and no longer forwards the task to other edge nodes [2], [4]. This is because in this fully connected network, the result of multiple

Authorized licensed use limited to: Hong Kong Polytechnic University. Downloaded on May 27,2024 at 10:28:24 UTC from IEEE Xplore. Restrictions apply.

forwarding of a task can be completed by one forwarding, and reducing the transmission time.

The system's reliability depends on the failure probability of each operational unit when processing tasks, such as edge nodes and transmission links. The failure of these components includes hardware, software, etc. Any failure case can be normalized to a Poisson distribution driven by a failure rate [28], [29], [30]. In this article, the processing time taken by edge node j to complete task $K_i(t)$ without failure is $T^c_{ij}(t)$. During the time interval $(0, T^c_{ij}(t)]$, failure occurs and is assumed to be a Poisson process with failure rate parameter α_j. The total number of failures that happen in the edge node j when processes task $K_i(t)$ is denoted by $N_j(T^c_{ij}(t))$. Therefore, the probability of $N_j(T^c_{ij}(t)) = k$ during the time interval $(0, T^c_{ij}(t)]$ can be computed as

$$Pr\{N_j(T^c_{ij}(t)) = k\} = \frac{(\alpha_j T^c_{ij}(t))^k}{k!} e^{-\alpha_j T^c_{ij}(t)}, k \geq 0, \quad (3)$$

where $k = 0$ if the task $K_i(t)$ is processed successfully during time interval $(0, T^c_{ij}(t)]$. Therefore, the reliability of the task $K_i(t)$ processed by the edge node j can be calculated as

$$R^c_{ij}(t) = e^{-T^c_{ij}(t)\alpha_j}, \quad (4)$$

We assume that the failure is a one-off incident that would not affect subsequent tasks [31]. Similarly, the reliability of the task $K_i(t)$ when transferred from edge node i to j can be calculated as

$$R^t_{ij}(t) = e^{-T^t_{ij}(t)\beta_{ij}}, \quad (5)$$

where β_{ij} denotes the failure rate of the transmission link between edge node i and edge node j.

B. Problem Definition

1) Decision Variable: In time slot t, when an edge node i has a task to process, it can either process it locally or offload it to other edge nodes. Binary decision variable $x_{ij}(t)$ denotes whether task $K_i(t)$ is offloaded to edge node j, where $x_{ij}(t) = 1$ if task $K_i(t)$ is offloaded to edge node j, otherwise $x_{ij}(t) = 0$. For generality, $x_{ii}(t) = 1$ denotes edge node i processes its arrival task locally.

2) Constraint: Each task is executed by only one edge node if a task arrives at edge node i in time slot t, which can be expressed as

$$\sum_{j=1}^{N} x_{ij}(t) \leq 1. \quad (6)$$

The completion time of task $K_i(t)$ can be calculated as

$$T_i(t) = \sum_{j \in \mathcal{N}} x_{ij}(t)(T^{tw}_{ij}(t) + T^t_{ij}(t) + T^{cw}_{ij}(t) + T^c_{ij}(t)), \quad (7)$$

where $T^{tw}_{ij}(t)$ and $T^t_{ij}(t)$ denote the transmission waiting time and transmission time from edge node i to j, respectively. $T^{cw}_{ij}(t)$ and $T^c_{ij}(t)$ denote the execution waiting time and execution time in edge node j, respectively. The task $K_i(t)$ should be completed before its deadline, which can be expressed as

$$T_i(t) \leq d_i(t). \quad (8)$$

Otherwise, the task is overdue and automatically dropped from the edge node.

3) Objective: The objective is to maximize the success rate of tasks in the duration time $\mathcal{T}$. A task $K_i(t)$ is successful if its completion time does not exceed its deadline $d_i(t)$ and executes successfully without any failure in execution or transmission. The objective is to maximize the task success rate defined as follows

$$\frac{|K_{\text{succ}}|}{|K_{\text{succ}}| + |K_{\text{drop}}| + |K_{\text{fail}}|}, \quad (9)$$

where $|K_{\text{succ}}|$ denotes the number of tasks processed successfully. $|K_{\text{drop}}|$ denotes the number of tasks that exceed their deadline and are dropped from the edge network, and $|K_{\text{fail}}|$ denotes the number of tasks that failed during execution or transmission.

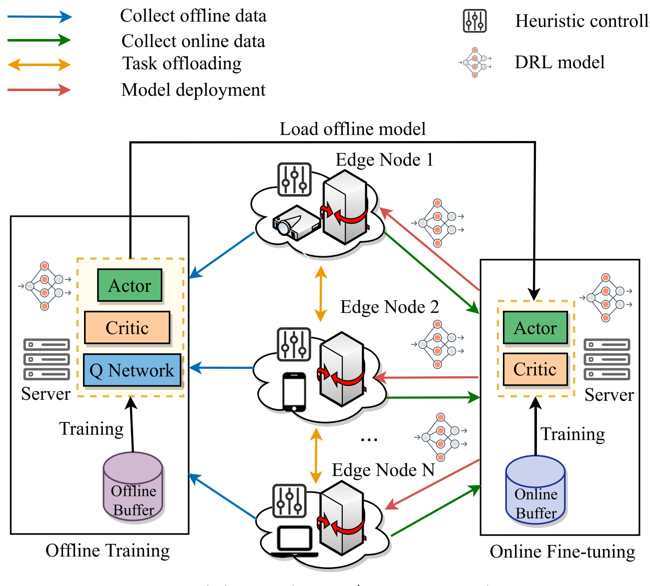


Fig. 2. **Overview of O2O-DRL.** O2O-DRL uses the task running logs generated by existing heuristic approaches to train an offline DRL model. Then, O2O-DRL fine-tunes the model using online logs when deploying the model in the real system.

IV. Proposed Offline-to-Online DRL Approach

This section presents a novel offline-to-online DRL approach for decentralized task offloading between cooperative edge nodes. We provide the overview of O2O-DRL in Fig. 2.

A. Reinforcement Learning

Reinforcement Learning (RL) involves an agent learning and optimizing its behavior through interaction with the environment to maximize cumulative rewards. In each interaction, the agent observes the environment state s_t, takes action a_t, and receives a reward r_t, leading to the next state s_{t+1}. Actions

Authorized licensed use limited to: Hong Kong Polytechnic University. Downloaded on May 27,2024 at 10:28:24 UTC from IEEE Xplore. Restrictions apply.

can be discrete or continuous, governed by stochastic policies $a_t = \pi(.|s_t)$ or deterministic policies $a_t = \mu(s_t)$. The agent aims to maximize the discounted cumulative reward $R_t = \sum_{t=0}^{\infty} \gamma^t r_t$, where $\gamma \in (0, 1)$. The interaction is modeled as a Markov Decision Process (MDP) defined by $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \rho_0 \rangle$. Here, $\mathcal{S}$ denotes the state space; $\mathcal{A}$ is the action space; $\mathcal{R} : S \times A \times S \rightarrow \mathbb{R}$ represents the reward function, with $r_t = R(s_t, a_t, s_{t+1})$; $\mathcal{P} : S \times A \rightarrow P(S)$ is the state transition function, where $P(s'|s, a)$ denotes the probability of transitioning to state s' given the current state s and action a; ρ_0 is the initial state distribution.

B. Dec-POMDP Formulation

A Dec-POMDP can be described as a tuple $\langle \mathcal{N}, \mathcal{S}, \mathcal{A}, P, r, \mathcal{O}, \rho, \gamma \rangle$. $\mathcal{N} = \{1, ..., N\}$ denotes the set of N agents. $s_t \in \mathcal{S}$ describes the state of the environment at time slot t. The initial state $s_0 \sim \rho$ is drawn from distribution ρ. At each time step t, each agent $i \in \mathcal{N}$ chooses an action $a_t^i \in \mathcal{A}$, forming a joint action $\boldsymbol{a_t} = \{a_t^i\}_{i=1}^N$. This causes a transition in the environment according to the state transition function $P(s_{t+1}|s_t, \boldsymbol{a_t})$. Then, all agents receive a scalar team reward $r_t = r(s_t, \boldsymbol{a_t})$. We consider a partially observable scenario in which each agent $i \in \mathcal{N}$ has individual observation $o_t^i \in \mathcal{O}$ and chooses its action with a decentralized policy $a_t^i \sim \pi^i(\cdot|o_t^i)$ that is based only on its observation. γ is the discounted factor of the return R_t, where $R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$.

For the decentralized task offloading problem, the agents correspond to the edge nodes, the observation of each edge node corresponds to the local network information, and the action of each edge node is the task offloading decision. The feedback of each task is a scalar value to denote whether a task is successful, failed, or dropped. The reward is the sum of feedback at the current time slot. Therefore, we establish the Dec-POMDP for our problem next.

1) Observation: At time slot t, each edge node i observes its local information from the edge network. such as the task arrival probability λ_i, the execution failure rate α_i, the computation capacity F_i, the length of execution queue l_i and receiving queues l_{ij}^r, the transmission rate r_{ij} to communicate with other edge nodes and the task information including task size $s_i(t)$, task complexity $c_i(t)$ and task deadline $d_i(t)$ when a task arrives. Thus, the observation of edge node i at time slot t can be defined as

$$o_t^i = \{\lambda_i, \alpha_i, F_i, \{l_i, l_{ij}^r\}, r_{ij}, \{s_i(t), c_i(t), d_i(t)\}\}.$$

Note that the observation includes different magnitudes of variables, so we let each element of the observation divide by the corresponding maximum value. This will make each element in the range of $[0, 1]$, ensuring that all elements are equally crucial for training the DRL model.

2) Action: At time slot t, if a task arrives, each edge node i chooses an action a_t^i to offload the task to edge node j, including itself, according to its current observation o_t^i. So each edge node at least needs N discrete action to represent all the edge nodes.

In this dynamic edge environment, an edge node may not receive a task from end-users in some time slots. However, the edge node also needs to select an action to form a transition $(o_t^i, a_t^i, r_t, o_{t+1}^i)$. Therefore, we add another action to the list of available actions to represent the case when an edge node does not receive a task from end-users. In this case, the edge node does not need to make the offloading decision and can only choose this action. This can be implemented by masking other available actions when an edge node does not need to make the offloading decision. Our proposed method can process random arrival tasks in the most realistic scenarios with this modification.

The neural network has a random policy in its initial training phase. It is helpful to guide the DRL model converging to a better solution by controlling the available actions. For example, if an edge node has a heavy workload, it will not want to receive the task from other edge nodes. Therefore, when an edge node needs to make a task offloading decision, it requests other edge nodes, whether their workload exceeds their execution queue length. Then, the edge node masks the available action to offload the task to those edge nodes whose workload has exceeded their queue length. This only needs 1 bit to send. Hence, the query of the workload state only incurs a small signaling overhead and can be ignored [25].

3) Reward: A task may be completed across multiple time slots. For task $K_i(t)$, edge node i will receive +1 feedback if task $K_i(t)$ is processed successfully in the later time slot. Otherwise, it will receive -1 feedback later if task $K_i(t)$ is dropped or failed. In this Dec-POMDP, all edge nodes share a team reward. The team reward in each time slot is defined as the sum of feedback of all edge nodes in this time slot.

C. Offline-to-Online DRL

Deploying models trained through DRL directly into operational environments, especially within the realms of the Internet of Vehicles (IoV) and robotics, presents considerable challenges [32]. The fundamental nature of DRL necessitates active engagement with the environment to gather data through a process known as exploration, which can inadvertently result in irreversible system faults. This exploration phase can escalate the costs of trial-and-error learning and degrade the user's QoE before the DRL models converge. However, heuristic approaches have been proposed and can be directly deployed in an edge computing environment without the training phase. We can deploy heuristic approaches in the edge networks and collect task running logs to train an offline DRL model. Then, when we deploy the DRL model in the edge network, we can collect new data by fine-tuning the DRL model. Therefore, our proposed approach consists of three phases, which are existing heuristic control, offline training, and online fine-tuning, and will be introduced next.

1) Heuristic Control: Heuristic-based task offloading approaches have generally been deployed in a real-world network system. However, these approaches have no learning mechanism, which cannot benefit from the history task running log. We can collect these data to train an offline DRL model. The DRL model can imitate the policy from heuristic approaches and improve its performance through the learning mechanism.

Authorized licensed use limited to: Hong Kong Polytechnic University. Downloaded on May 27,2024 at 10:28:24 UTC from IEEE Xplore. Restrictions apply.

The benefit of using the task running log of traditional heuristic approaches to train the DRL model can be summarized in the following two aspects. First, we can warm-start the DRL model with a good policy instead of a random policy in the initial online phase. A random policy may significantly affect users' QoE. Second, the heuristic policy in the task running log can guide the DRL model to converge to a better solution. We would show it in the experiment part.

2) Offline Training: It is intuitive to train N DRL models for each edge node using individual task running logs. However, the network information in each agent's observation is limited. Each edge node cannot aware of the task offloading decision made by other edge nodes. It is challenging for all the agents to converge to a good solution. To address this problem, we can utilize the task running logs of all edge nodes to train a sharing DRL model and deploy the sharing DRL model in each edge node for decentralized task offloading. However, the definition of observation in Section IV-B1 is insufficient for the neural network to discriminate each edge node's observation. Therefore, we modified the observation by combining it with a one-hot encoding of the edge node index. For example, the one-hot encoding of edge node i is an array with length N, where the index $i-1$ is 1 while other are 0. Since the training is an offline process, we can train the offline DRL model in a powerful edge node or in the cloud server. After training, we can deploy the DRL model in each edge node for decentralized task offloading.

In order to train an offline DRL model using the task running logs of heuristic approaches, we use off-policy DRL in the offline phase. DQN [33] is the most classical off-policy DRL method and has been widely applied in edge computing. However, DQN uses a ϵ-greedy policy, which cannot explore a good solution to utilize the computation resources of all edge nodes in our online decentralized task offloading problem while action random choices based on the probability of each action can utilize the computation resource of all edge nodes. Therefore, we design offline training of the DRL model based on Discrete SAC [34], [35].

Two Q networks are used to avoid value overestimation. Let ϕ_1 and ϕ_2 be the parameters of two Q networks, respectively. Correspondingly, $\bar{\phi_1}$ and $\bar{\phi_2}$ be the target network parameters of ϕ_1 and ϕ_2, respectively. θ denotes the parameters of the actor network. The soft value function can be calculated as follows

$$V(o_{t+1}^i) = \pi_\theta(o_{t+1}^i)^T \left(\min_{i=1,2} \bar{Q}_{\phi_i}(o_{t+1}^i) - \alpha \log \pi_\theta(o_{t+1}^i) \right).$$

Then, the target value can be calculated as

$$y(r_t, o_{t+1}^i) = r_t + \gamma V(o_{t+1}^i).$$

we can update each Q network by minimizing the following Mean Square Error (MSE) using gradient descent:

$$\nabla_{\phi_i} \frac{1}{|\mathcal{B}|} \sum_{\mathcal{B}} \sum_{i=1}^{N} (Q_{\phi_i}(o_t^i, a_t^i) - y(r_t, o_{t+1}^i))^2.$$

However, there exists a distributional shift between the policy of heuristic approaches that collected the data and the offline learning policy [14]. To address the problem, we adopt the CQL regularizer [15] to optimize the Q network. The regularizer can be expressed as

$$\mathcal{R}(o_t^i, a_t^i) = \log \sum_a \exp(Q_{\phi_i}(o_t^i, a)) - Q_{\phi_i}(o_t^i, a_t^i).$$

Note that we need to minimize the regularizer. Therefore, the first term is to decrease the Q values for unseen actions and the second term is to increase the Q values for seen actions in the offline data. With the regularizer, we update the Q network by minimizing the following equation using gradient descent:

$$\nabla_{\phi_i} \frac{1}{|\mathcal{B}|} \sum_{\mathcal{B}} \sum_{i=1}^{N} (Q_{\phi_i}(o_t^i, a_t^i) - y(r_t, o_{t+1}^i))^2 + \lambda_c \mathcal{R}(o_t^i, a_t^i), \tag{10}$$

where λ_c is the weight of the regularizer.

For the actor update, we need to maximize the soft value function using gradient ascent:

$$\nabla_\theta \frac{1}{|\mathcal{B}|} \sum_{\mathcal{B}} \sum_{i=1}^{N} V(o_{t+1}^i). \tag{11}$$

Importantly, in order to fine-tune the DRL model based on online data generated by the DRL scheduler, we can train the model with on-policy actor-critic methods. Therefore, we train an additional critic network in the offline phase. Let ω be the parameters of the critic network. We update the critic parameters by minimizing the following MSE objective using gradient descent:

$$\nabla_\omega \frac{1}{|\mathcal{B}|} \sum_{\mathcal{B}} \sum_{i=1}^{N} (V_\omega(o_t^i) - y(r_t, o_{t+1}^i))^2. \tag{12}$$

Notes that the critic network will be used in the online phase.

The reward value in each step has a wide range, which seriously affects the learning of neural networks. Therefore, we normalize the rewards in the offline task running logs to improve convergence speed and performance. We provide the algorithm to train the offline DRL model in Algorithm 1.

3) Online Fine-Tuning: Although we can directly deploy the offline DRL model in the system for online task offloading, we can further improve its performance via online fine-turning. When deploying the DRL model in the system, each edge node will accumulate new task running logs during execution. Therefore, we can periodically collect the real time data in all edge nodes to fine-tune the DRL model and then update the DRL model in each edge node. Note that we only need to deploy the actor network in each edge node for online task offloading. The actor network only uses three fully connected layers to build the neural network. Therefore, the data size of the actor model is less than 100KB, which only causes a small communication overhead. Moreover, we only update the model when collecting enough data to ensure the adjustments are based on sufficient information and avoid overfitting or frequent model changes.

The challenge that prevents online optimization is the distribution shift between offline data and online data [16]. The Q network and critic network is pre-trained based on offline data, which will cause overestimation for unseen observations in the online phase. Notes that we have trained an additional critic

Authorized licensed use limited to: Hong Kong Polytechnic University. Downloaded on May 27,2024 at 10:28:24 UTC from IEEE Xplore. Restrictions apply.

Algorithm 1: Offline Training

input : Initialize Q network Q_{ϕ_1}, Q_{ϕ_2}; actor network π_θ; critic network V_ω.
Initialize target Q network $\bar{Q}_{\phi_1} \leftarrow Q_{\phi_1}$, $\bar{Q}_{\phi_2} \leftarrow Q_{\phi_2}$.
Load offline data $\mathcal{D}$.
Normalize the rewards r_t in data $\mathcal{D}$.

1 **for** $t \leftarrow 1$ **to** N **do**
2 Sample min-batch B from $\mathcal{D}$.
3 Train the critic network V_ω using gradient descent from equation (12).
4 Train the actor network π_θ using gradient ascent from equation (11).
5 Train the Q_{ϕ_1} and Q_{ϕ_2} network using gradient descent from equation (10).
6 Update the target Q network $\bar{Q}_{\phi_1}$, $\bar{Q}_{\phi_2}$ with
7 $\bar{\phi}_i = (1-\tau)\bar{\phi}_i + \tau\phi_i$ for i = 1, 2.
8 **end**

output: actor π_θ, critic V_ω.

Algorithm 2: Online Fine-Tuning

input : load offline actor π_θ and critic V_ω.

1 **while** *True* **do**
2 Collect a set of trajectories $\mathcal{D}_k$ by running policy π_θ in the edge network.
3 Normalized the rewards r_t in $\mathcal{D}_k$.
4 Compute advantage A_t^i.
5 **for** $k \leftarrow 1$ **to** K *epochs* **do**
6 Update the actor network using gradient ascent by equation (16).
7 Update the critic network using gradient descent by equation (19).
8 **end**
9 **end**

output: actor π_θ, critic V_ω.

network in the offline phase. Therefore, we can use on-policy actor-critic DRL to fine-tune the DRL model.

To address this issue, we leverage the GAE in [36] to balance the bias of value estimation and variance of return. The experiment in Section V shows that our approaches can address the overestimation problem and avoid performance degradation in the initial online fine-tuning phase.

We can calculate the GAE recursively as

$$A_t^i = \delta_t^i + (\lambda\gamma)A_{t+1}^i, \tag{13}$$

$$\delta_t^i = r_t + \gamma V_{\omega_{\text{old}}}(o_{t+1}^i) - V_{\omega_{\text{old}}}(o_t^i), \tag{14}$$

where δ_t^i is the Temporal Difference (TD) error. Therefore, the target value can be calculated as follows,

$$V_t^i = A_t^i + V_{\omega_{\text{old}}}(o_t^i). \tag{15}$$

We use clip trick in PPO [37] to update the actor network. Therefore, we update the actor network by maximizing the following objective using gradient ascent:

$$\nabla_\theta \frac{1}{|\mathcal{D}|T} \sum_{\mathcal{D}} \sum_{t=0}^{T} \sum_{i=1}^{N} \left[\min\left(r(\theta)A_t^i, \eta(r(\theta))A_t^i\right)\right], \tag{16}$$

where $r(\theta)$ is the ratio between the new policy and the old policy:

$$r(\theta) = \frac{\pi_\theta(a_t^i|o_t^i)}{\pi_{\theta_{\text{old}}}(a_t^i|o_t^i)} \tag{17}$$

and $\eta(x)$ is a clip function expressed as

$$\eta(x) = \begin{cases} 1+\varepsilon, & x > 1+\varepsilon \\ x, & 1-\varepsilon \le x \le 1+\varepsilon \\ 1-\varepsilon, & x < 1-\varepsilon \end{cases}. \tag{18}$$

We update the critic network by minimizing the following MSE objective using gradient descent:

$$\nabla_\omega \frac{1}{|\mathcal{D}|T} \sum_{\mathcal{D}} \sum_{t=0}^{T} \sum_{i=1}^{N} \left(V_\omega(o_t^i) - V_t^i\right)^2. \tag{19}$$

Similar to offline training, we normalized the rewards in collected trajectories to improve the DRL convergence speed and performance. We provide the algorithm for online fine-tuning in Algorithm 2.

V. Evaluation

In this section, we first present the setup and benchmarks in our simulation. Then we use simulation environments to evaluate the performance of our proposed approach for solving the decentralized task offloading problem.

A. Simulation Setup

We use *Python* to implement a simulator for the edge computing environment. The default parameter of the environment is as follows. The number of edge nodes is 20. The total number of time slots is 100, and each time slot is 0.5s. The task arrival probability at each edge node in each time slot is uniformly distributed on [0, 1]. The input size of tasks follows a uniform distribution on [1000, 8000]KB. The complexity of tasks is uniformly distributed on [800, 2400] cycles/bit [13]. The deadline for the task is 4s. The number of CPU cores in the edge node is randomly selected from $\{4, 8, 16, 20, 24, 28, 32\}$ [3], and the CPU frequency of each core is 3GHz [3]. The transmission rate among edge nodes follows uniform distribution on [10, 40]MB/s. The failure rate of edge nodes is uniformly distributed on [0, 0.1]. The failure rate of transmission links is uniformly distributed on [0, 0.03] [28].

We use the *Pytorch* machine learning framework to build and train the DRL model. The number of hidden layers in the Q, critic, and actor networks is 2. The dimension of the hidden layer is 64. The activation function in the above networks is the *relu* function. The learning rate of the above networks is

Authorized licensed use limited to: Hong Kong Polytechnic University. Downloaded on May 27,2024 at 10:28:24 UTC from IEEE Xplore. Restrictions apply.

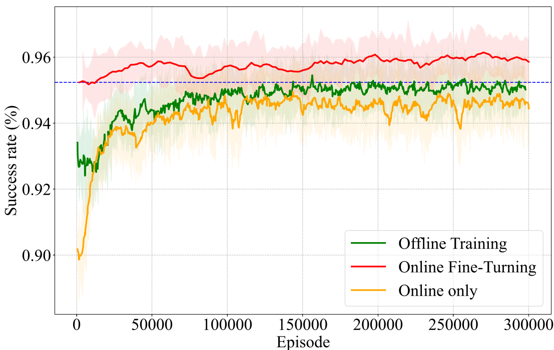


Fig. 3. The learning curve of task success rate.

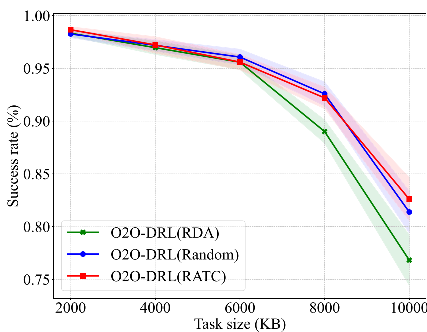


Fig. 4. Success rate of tasks under different heuristic approaches.

0.0003. The discount factor γ of the reward is 0.99. The soft update parameter τ in the Q network is 0.0005. The buffer size and batch size in offline training are 10^6, 128, respectively. The α is 0.05. The weight λ_c of the regularizer is 0.1. The λ is 0.95. The clip parameter ϵ is 0.2. The epochs K in online training is 4.

B. Baseline Approaches

We compare our proposed O2O-DRL approach against the following approaches.

- **Reliability-Aware Two Choices (RATC).** Inspired by the "power of two choices" in distributed task scheduling [10], [21], [38], we proposed this distributed heuristic approach considering reliability. Specifically, when an edge node receives a task, it randomly probes two edge nodes. If both edge nodes can process the task within its deadline, it selects the more reliable one. Otherwise, it selects the edge node to process the task with lower latency.
- **Reactive Distributed Algorithm (RDA) [9].** When a task arrives in an edge node, it processes the task locally if the length of its execution queue is below the max queue length. Otherwise, it broadcasts the information to other edge nodes for task offloading. Then it waits for two edge nodes that respond to it and selects the edge node that can process the task with a lower delay.
- **Offline only.** It only uses the offline training in O2O-DRL without further fine-tuning.
- **Online only.** It only uses the online training method of O2O-DRL without warming up the neural network models with offline training.

To evaluate each method's performance, we run 10 episodes in each edge network setting and draw the mean and standard deviation of the task success rate of each method in the figures.

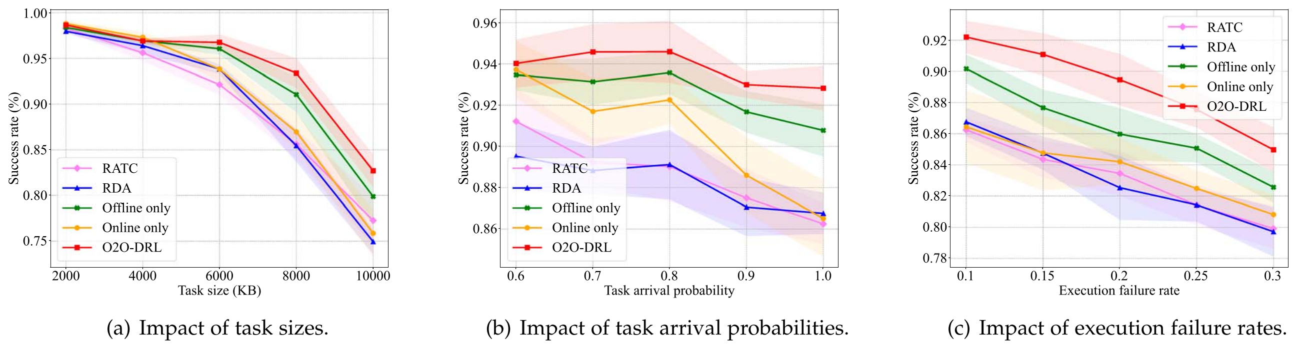
C. Convergence of Proposed Method

Fig. 3 shows the learning curves of the task success rate of O2O-DRL compared with Online only. O2O-DRL includes two phases: offline training and online fine-tuning. The experiment results show that O2O-DRL can converge to a better solution by using the task running logs of the heuristic approach to warm up the neural network parameters. In the offline phase, O2O-DRL uses the task running logs of the traditional decentralized approach, RATC, to train the DRL model without interaction with the network system. O2O-DRL can avoid converging to a sub-optimal solution compared with directly online learning. The heuristic policy, RATC, can guide the DRL model to explore a better solution. We notice that our approach can further improve the performance of the DRL model in the online fine-tuning and avoid performance degradation in the initial online phase. Notably, the stochastic generation of tasks may lead to slight fluctuations in the success rate between the starting point of online fine-tuning and the termination point of offline training, as indicated by the blue horizontal dashed line in Fig. 3, consistent with our expected outcomes. Furthermore, because we use on-policy DRL to fine-tune the DRL model in an actor-critic style, we can leverage GAE to balance the bias and variance for unseen observations in the online phase.

D. Impact of the Heuristic Approaches

O2O-DRL uses the task running logs of traditional heuristic approaches to train an offline DRL model. Then, it uses the online data to improve its performance when deploying the model in the online system. Therefore, different heuristic approaches will affect the performance of O2O-DRL. Consequently, we evaluate the performance of O2O-DRL under different heuristic approaches, as shown in Fig. 4. The results show that O2O-DRL has a higher success rate of tasks based on the task running logs of RATC compared with building on other heuristic approaches. When an edge node deploys RDA for task offloading, it will process tasks locally if its queue length is under the max queue length. As a result, O2O-DRL may learn to imitate the RDA to let more tasks be processed locally, leading to sub-optimal performance. The RATC considers both latency and reliability, which can guide the DRL model to explore a better solution. We notice that O2O-DRL even can use the task running logs of the poorest performance approach, Random, to learn a relatively good policy. Therefore, O2O-DRL is robust to existing heuristic approaches and can build on these approaches to improve its performance via the learning mechanism.

Authorized licensed use limited to: Hong Kong Polytechnic University. Downloaded on May 27,2024 at 10:28:24 UTC from IEEE Xplore. Restrictions apply.



(a) Impact of task sizes.

(b) Impact of task arrival probabilities.

(c) Impact of execution failure rates.

Fig. 5. Success rate of tasks under different: (a) task sizes; (b) task arrival probabilities; (c) task execution failure rates.

E. Impact of Task Sizes

We compare O2O-DRL with other methods under different task sizes, as shown in Fig. 5(a). The results show that O2O-DRL has a significant advantage over other methods. Herein, the task size refers to the maximum size of tasks generated in the network.

We notice that when the task size becomes large, the success rate of all approaches becomes low. The reason can be divided into two folders. First, as the task size becomes large, the transmission time and execution time of each task become longer, prolonging the task completion time. Therefore, more tasks will be over their deadline and will be dropped from the network system. Second, as the transmission time and execution time increase, the reliability of tasks in transmission and execution will decrease, and more tasks will fail. When the task size is uniformly distributed in [1000, 10000]KB, the success rate of all methods drops significantly. This is because all edge nodes' computation and communication resources are limited and can not satisfy all the computation-intensive and delay-sensitive tasks. Moreover, the *online only* exhibits inferior performance compared to the RATC due to the influence of low-quality early-stage data, leading the learning process astray and resulting in the agent being ensnared in a locally optimal solution.

F. Impact of Task Arrival Probabilities

To investigate the task arrival probability that influences the performance of our proposed methods, we vary it and test the task success rate. The simulation results are shown in Fig. 5(b), where the task arrival probabilities range from [0,0.6] to [0, 1]. The results show that O2O-DRL outperforms other approaches at all the task arrival probabilities. We find that the success rate decreases as the task arrival probability increases. This is because the same edge nodes can provide limited computation resources and can not afford large concurrent requests.

The performance of Online only drops significantly as the task arrival probability increases. This is because directly online training the DRL model without other heuristic knowledge will converge to a sub-optimal solution. Moreover, as more tasks arrive in each time slot, it is hard for Online only to explore a good policy starting from a random policy in the initial online learning, which will trap into local optimization. Compared with Online only, the task running logs of RATC can guide Offline only and O2O-DRL to converge to a better solution. When the task arrival probability is uniformly distributed in [0, 0.6], RATC can offload more tasks to reliable edge nodes to avoid execution failure and performs better than RDA, which only focuses on latency without considering reliability. On the other hand, when the task arrival probability is uniformly distributed in [0, 1.0], the workload of all the edge nodes is heavy, and the delay becomes the critical factor affecting the task success rate. Therefore, the task offloading policy of RATC will induce additional transmission overhead and has relatively poor performance compared with RDA.

G. Impact of Execution Failure Rates

The execution failure rate denotes the probability that a task will fail during one period. Fig. 5(c) presents the experiment results with a varied execution failure rate of edge nodes which from [0, 0.1] to [0, 0.3]. The results show that our approach always outperforms other benchmarks, which illustrates the advantage of the O2O-DRL in improving the success rate of tasks. We can observe that the success rate of tasks for all methods decreases with the increased execution failure rate of edge nodes. That is because more tasks will fail as the execution failure rate of edge nodes increase. DRL-based methods can trade-off between delay and reliability via the design of reward function. Since an edge node will receive -1 feedback when its task is failed or is dropped and receive +1 feedback when it is successful. These methods can balance the latency and reliability for a higher success rate. However, O2O-DRL outperforms Online-only and Offline-only in all execution failure rates due to the heuristic guide of RATC and online fine-tuning.

H. Impact of the Number of Edge Nodes

We evaluate the performance of our approach with a different number of edge nodes in the system, as shown in Fig. 6. We can find that O2O-DRL performs better than other approaches in all test numbers of edge nodes. The results show that O2O-DRL is scalable to a large number of edge nodes. The performance of all methods is sensible to the configuration of edge nodes and task arrival probability in each edge node. Online only trains the DRL model directly by online learning in the edge network. Therefore, it will fall into a sub-optimal solution

Authorized licensed use limited to: Hong Kong Polytechnic University. Downloaded on May 27,2024 at 10:28:24 UTC from IEEE Xplore. Restrictions apply.

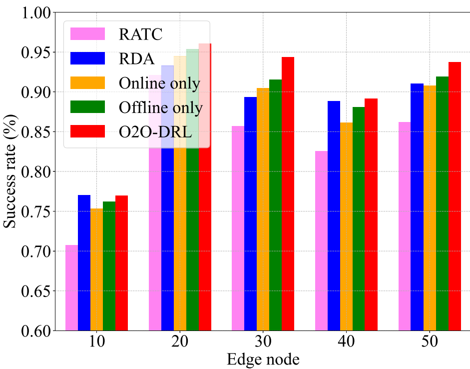


Fig. 6. Success rate of tasks under the different number of edge nodes.

without a heuristic guide. Offline only uses the task running logs of RATC to train a DRL model and achieve a relatively good performance. However, O2O-DRL can further improve its performance by online fine-tuning.

VI. Experiment in Kubernetes-Based Testbed

To validate the practicality and applicability of the O2O-DRL approach, we build a testbed based on *Kubernetes*. The *Python* simulator used in Section V can quickly verify our idea. However, since each edge node is just an instance in the program, we can not deploy the real application in the edge node to test it. Moreover, the communication between edge nodes is implemented within the program without real communication based on TCP/IP protocol. Using containerization technology, we can create multiple emulated edge nodes with isolated computing space and network space in a single computing server. These emulated nodes can run applications and communicate with each other through networks.

A. Testbed

Fig. 7 depicts a high-level overview of the architecture of our designed testbed. The architecture comprises *DRL Layer*, *Control Layer*, *Execution Layer*, and the *Middleware*. All the components run on top of the *Kubernetes*. The *Container Runtime* we used in *Kubernetes* is the *Docker Engine*. Therefore, we can use the *Dockerfile* to build the image of *Trainer*, *Controller* and *Edge Node*. The testbed is implemented via *Java* language. The source code of the testbed has about 5000 lines of code. It is open-sourced on GitHub via the link: https://github.com/lhc0512/edge-computing.

The top layer is the *DRL Layer*. This layer is responsible for training the DRL models. The *Trainer* runs as a *Pod* in the *Kubernetes*. We provide the *DRL algorithm* implementation based on *Deep Java Library (DJL)*, which is a deep learning framework for *Java*. The *DRL algorithm* consists of three parts, which include *Model*, *Buffer*, and *Agent*. The *Model* contains the neural network of DRL models. The *Buffer* stores the transitions containing observations, actions, and rewards. These transitions are reconstructed from the task running logs stored in the *Database*. The *Agent* provides two main functional interfaces, *predict* and *train*. The *predict* is responsible for the

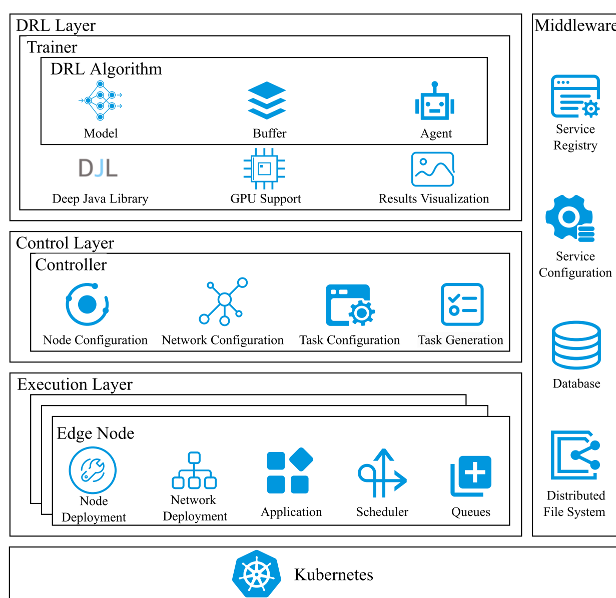


Fig. 7. **The architecture of testbed.** The testbed consists of *DRL Layer*, *Control Layer*, *Execution Layer*, and the *Middleware*. All the components run on top of the *Kubernetes*.

Agent to select an action based on current observation, and the *train* is to train the DRL *Model* using the *Buffer*. We build the image of *Trainer* based on *nvidia*'s docker images to train the DRL model in the container using GPU. We provide the *Results Visualization* to help users conduct tests. The models, buffers, and experiment results are stored in the *Distributed File System (DFS)*.

Below the *DRL Layer* is the *Control Layer*. The Controller runs as a *Pod* in the *Kubernetes*. The *Controller* acts as a coordinator between users and *Edge Nodes*. First, the users can write the configuration files in the *Service Configuration*. Then the *Node Configuration* and *Network Configuration* of the *Controller* are responsible for parsing the configuration file into specific deployment. Next, the *Controller* writes the deployment information into the *Database*. The responsibility of *Task Configuration* is to parse the configuration file in *Service Configuration*. Then, The *Task Generation* acts as the end devices to generate tasks and send these tasks to *Edge Nodes* for task offloading. Overall, the Controller functions as a virtual rather than a centralized scheduling node, responsible for configuring the network environment and generating tasks. It abstains from engaging in task offloading decisions, leaving the responsibility of task scheduling autonomously managed by individual nodes.

The bottom layer is the *Execution Layer*. Each *Edge Node* runs as a *Pod* in the *Kubernetes*. When setting up the *Edge Nodes*, the *Network Deployment* and the *Node Deployment* are responsible for reading the network and node information from the *Database* to deploy, respectively. The *Queues* in each edge node include an execution queue to execute the tasks and $N-1$ sending queues and $N-1$ receiving queues to communicate with other edge nodes. The *Application* denotes the type of application that each edge node runs. The *Scheduler* module

Authorized licensed use limited to: Hong Kong Polytechnic University. Downloaded on May 27,2024 at 10:28:24 UTC from IEEE Xplore. Restrictions apply.

of each *Edge Node* is responsible for deciding where and when to offload the computational tasks within the *Application*. The *Scheduler* can be a heuristic or DRL scheduler. A DRL scheduler needs to load the actor model from the DFS.

The *Middleware* includes *Service Registry*, *Service Configuration*, *Database*, and *Distributed File System (DFS)*. The *Trainer*, *Controller*, and *Edge Node* are registered as services in the *Service Registry*. The *Service Registry* stored the IP and port of each service. Then, each service can discover other services and communication with them via querying from the service list in the *Service Registry*. To detect the health of each service, the registry maintains communication with the services through a heartbeat mechanism. We use *Nacos* as the implementation of *Service Registry*. The configuration files of *Trainer*, *Controller*, and *Edge Nodes* are stored in *Service Configuration*. When each service starts, it reads the configuration file from the *Service Configuration* and configures each component based on the configuration file. When the configuration files have been modified, the *Service Configuration* can send the changed content to refresh the corresponding service. We also use *Nacos* as the implementation of *Service Configuration*. The metadata of task running logs is stored in the *Database*. This information includes task id, time slot, source of the task, execution place of the task, running status, task size, CPU cycle, deadline, transmission time, transmission waiting time, execution time, and execution waiting time. We use *MySQL* as the *Database*. The DFS is responsible for storing files such as DRL models, DRL buffers, and figures of experiment results. For example, the *Trainer* can upload its DRL models in the DFS, and *Edge Nodes* can download the DRL models from the DFS for task offloading. The *Trainer* can upload the buffers constructed from the task logs in *Database*. In addition, the *Trainer* can upload the figures of training information to the DFS. We use HDFS as the DFS.

B. Deployment of O2O-DRL on the Testbed

O2O-DRL consists of three parts: data collection by heuristic approaches, offline training, and online fine-tuning. For the first part, we can select the heuristic *Scheduler* in the *Edge Node*. During the execution, the task running logs will be stored in the *Database*. After collecting enough task running records, we use these logs to construct the transitions of DRL, which include observations, actions, and rewards. Then, we stored these transitions in DFS for offline training.

In the offline training phase, we select the offline *DRL Algorithm* in *Trainer* and load the transitions from DFS into *Buffer*. Then, we can train an offline DRL model and store the model in DFS. Finally, the edge nodes can load the actor model from the DFS into the *Scheduler* for online task offloading.

In the online fine-tuning phase, we can select the online *DRL Algorithm* in the *Trainer*. Then, we load the offline DRL model from the DFS for online task offloading. During the execution, the edge nodes will record new task running logs into the *Database*. After some time, we can read the logs to reconstruct the transitions of DRL. Then, load these transitions into the *Buffer* and fine-tune the DRL model using online *DRL Algorithm*. Then, we can store the updated DRL model in the DFS and notify all the edge nodes to update their actor model through *Controller*.

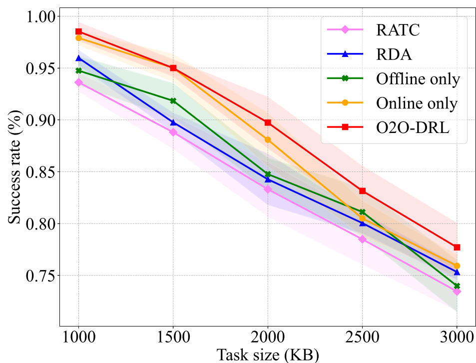


Fig. 8. Success rate of tasks under different task sizes.

C. Experiment Results

Furthermore, we evaluated the performance of O2O-DRL in this testbed. The testbed runs on a server. The CPU configuration of the server is the AMD Ryzen Threadripper 3990X, which has 64 cores and 128 threads. The memory of the server is 256GB. The number of edge nodes is 10. To verify our methods quickly in this testbed, we focus on task offloading for small tasks. The duration of the time slot is 0.1s. The total time slots are 50. The deadline for tasks is 1s. The buffer size in offline training is 10000, which needs 200 episodes to collect heuristic data. The number of episodes for online fine-tuning is 50. We compare O2O-DRL with other methods under different task sizes. The results are shown in Fig. 8. We can find that O2O-DRL outperforms other approaches under all the different task sizes. On the other hand, we can observe that the success rate of tasks for all methods decreases with the increase of task sizes. The reason is as follows: As the input data of a task increases, the transmission time and computation time of each task become longer. However, the computing resources of all edge nodes are limited, and more tasks will be dropped when the task completion is over the deadline. At the same time, the reliability of tasks during transmission and execution decreases, resulting in more task failures. The overall manifestation is a decrease in the success rate of the task. Compared with the simulation in Section V-E, the offline buffer is 10000 and less than 10^6. However, Offline-only can still converge to a good solution. Likewise, Online-only and O2O-DRL can converge into 50 episodes. In short, the simulation environment in Section V can quickly verify our idea, while the testbed can validate the practicality and applicability of the O2O-DRL.

VII. Conclusion

This paper proposed an O2O-DRL approach for task offloading in the edge network with decentralized execution. To maximize the success rate of tasks from a long-term perspective, we first defined the decentralized task offloading problem by

Authorized licensed use limited to: Hong Kong Polytechnic University. Downloaded on May 27,2024 at 10:28:24 UTC from IEEE Xplore. Restrictions apply.

considering task delay and reliability. Then, we transformed the formulated problem into a Dec-POMDP. To alleviate the cold-start problem caused by DRL, we leveraged the task running logs from traditional heuristic approaches to warm-start the online DRL. Moreover, we can further improve the DRL model's performance by using on-policy actor-critic DRL to fine-tune the DRL model when deploying the model into the edge network. Extensive experimental results show that our approach can significantly improve the task success rate and avoid the cold-start problem compared to alternative baselines.

In this study, our emphasis has been on single-hop networks featuring indivisible tasks. As a future work, we intend to broaden this scope to multi-hop networks, integrating task offloading decisions with network routing selection. The task types could also be expanded to encompass Service Function Chain (SFC) or tasks modeled as directed acyclic graphs (DAG), such as distributed training and inference of large language models. This will enhance the theoretical foundation of our research and aid in the practical application of edge computing, particularly in scenarios with high task complexity and dynamic network conditions, tackling the complex challenges of modern networked computing environments.

References

[1] C. Sun, X. Wu, X. Li, Q. Fan, J. Wen, and V. C. Leung, "Cooperative computation offloading for multi-access edge computing in 6G mobile networks via soft actor critic," *IEEE Trans. Netw. Sci. Eng.*, early access, 2021.

[2] X. Wang, Z. Ning, and S. Guo, "Multi-agent imitation learning for pervasive edge computing: A decentralized computation offloading algorithm," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 2, pp. 411–425, Feb. 2021.

[3] X. Qiu, W. Zhang, W. Chen, and Z. Zheng, "Distributed and collective deep reinforcement learning for computation offloading: A practical perspective," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 5, pp. 1085–1101, May 2021.

[4] S. Josilo and G. Dán, "Decentralized algorithm for randomized task allocation in fog computing systems," *IEEE/ACM Trans. Netw.*, vol. 27, no. 1, pp. 85–97, Feb. 2019.

[5] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1619–1632, Aug. 2018.

[6] X. He and S. Wang, "Peer offloading in mobile-edge computing with worst case response time guarantees," *IEEE Internet Things J.*, vol. 8, no. 4, pp. 2722–2735, Feb. 2021.

[7] Y. Li, X. Wang, X. Gan, H. Jin, L. Fu, and X. Wang, "Learning-aided computation offloading for trusted collaborative mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 19, no. 12, pp. 2833–2849, Dec. 2020.

[8] Y. Xiao and M. Krunz, "QoE and power efficiency tradeoff for fog computing networks with fog node cooperation," in *Proc. IEEE Conf. Comput. Commun.*, Piscataway, NJ, USA: IEEE Press, 2017, pp. 1–9.

[9] S. Sthapit, J. Thompson, N. M. Robertson, and J. R. Hopgood, "Computational load balancing on the edge in absence of cloud and fog," *IEEE Trans. Mobile Comput.*, vol. 18, no. 7, pp. 1499–1512, Jul. 2019.

[10] L. Lin, P. Li, J. Xiong, and M. Lin, "Distributed and application-aware task scheduling in edge-clouds," in *Proc. Int. Conf. Mobile Ad-Hoc Sensor Netw.*, Piscataway, NJ, USA: IEEE Press, 2018, pp. 165–170.

[11] X. Wang, Y. Han, V. C. M. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 2, pp. 869–904, Secondquarter 2020.

[12] T. Chu, S. Chinchali, and S. Katti, "Multi-agent reinforcement learning for networked system control," in *Proc. Int. Conf. Learn. Representations*, 2020, *arXiv:2004.01339*.

[13] C. Liu, F. Tang, Y. Hu, K. Li, Z. Tang, and K. Li, "Distributed task migration optimization in MEC by extending multi-agent deep reinforcement learning approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 7, pp. 1603–1614, 2021.

[14] S. Fujimoto, D. Meger, and D. Precup, "Off-policy deep reinforcement learning without exploration," in *Proc. 36th Int. Conf. Mach. Learn.*, vol. 97, 2019, pp. 2052–2062.

[15] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative Q-learning for offline reinforcement learning," *Adv. Neural Inf. Proc. Syst.*, Vol. 33, pp. 1179–1191, 2020.

[16] S. Lee, Y. Seo, K. Lee, P. Abbeel, and J. Shin, "Offline-to-online reinforcement learning via balanced replay and pessimistic Q-ensemble," in *Proc. Conf. Robot Learn.*, vol. 164, 2021, pp. 1702–1712.

[17] Y. Sahni, J. Cao, L. Yang, and Y. Ji, "Multi-hop multi-task partial computation offloading in collaborative edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 5, pp. 1133–1145, May 2021.

[18] X. He and S. Wang, "Peer offloading in mobile-edge computing with worst case response time guarantees," *IEEE Internet Things J.*, vol. 8, no. 4, pp. 2722–2735, Feb. 2021.

[19] M. Yang, H. Zhu, H. Qian, Y. Koucheryavy, K. E. Samouylov, and H. Wang, "Peer offloading with delayed feedback in fog networks," *IEEE Internet Things J.*, vol. 8, no. 17, pp. 13690–13702, Sep. 2021.

[20] R. Beraldi, C. Canali, R. Lancellotti, and G. P. Mattia, "Distributed load balancing for heterogeneous fog computing infrastructures in smart cities," *Pervasive Mobile Comput.*, vol. 67, Sep. 2020, Art. no. 101221.

[21] R. Beraldi and G. P. Mattia, "Power of random choices made efficient for fog computing," *IEEE Trans. Cloud Comput.*, vol. 10, no. 2, pp. 1130–1141, Apr.–Jun. 2022.

[22] G. Mitsis, E. E. Tsiropoulou, and S. Papavassiliou, "Price and risk awareness for data offloading decision-making in edge computing systems," *IEEE Syst. J.*, vol. 16, no. 4, pp. 6546–6557, Dec. 2022.

[23] Z. Cheng, M. Min, M. Liwang, L. Huang, and Z. Gao, "Multiagent DDPG-based joint task partitioning and power control in fog computing networks," *IEEE Internet Things J.*, vol. 9, no. 1, pp. 104–116, Jan. 2022.

[24] J. Baek and G. Kaddoum, "Heterogeneous task offloading and resource allocations via deep recurrent reinforcement learning in partial observable multifog networks," *IEEE Internet Things J.*, vol. 8, no. 2, pp. 1041–1056, Jan. 2021.

[25] M. Tang and V. W. S. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Trans. Mobile Comput.*, vol. 21, no. 6, pp. 1985–1997, Jun. 2022.

[26] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Piscataway, NJ, USA: IEEE Press, 2016, pp. 1451–1455.

[27] Z. Hong, W. Chen, H. Huang, S. Guo, and Z. Zheng, "Multi-hop cooperative computation offloading for industrial IoT-edge-cloud computing environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 12, pp. 2759–2774, Dec. 2019.

[28] X. Hou et al., "Reliable computation offloading for edge-computing-enabled software-defined IoV," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 7097–7111, Aug. 2020.

[29] X. Hou, Z. Ren, J. Wang, S. Zheng, and H. Zhang, "Latency and reliability oriented collaborative optimization for multi-UAV aided mobile edge computing system," in *Proc. IEEE Conf. Comput. Commun. Workshops*, Piscataway, NJ, USA: IEEE Press, 2020, pp. 150–156.

[30] J. Liu et al., "Reliability-enhanced task offloading in mobile edge computing environments," *IEEE Internet Things J.*, vol. 9, no. 13, pp. 10382–10396, Jul. 2022.

[31] J. Jia, L. Yang, and J. Cao, "Reliability-aware dynamic service chain scheduling in 5G networks based on reinforcement learning," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Piscataway, NJ, USA: IEEE Press, 2021, pp. 1–10.

[32] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: A survey," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, 2020, pp. 737–744.

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

[34] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. 35th Int. Conf. Mach. Learn.*, vol. 80, 2018, pp. 1856–1865.

Authorized licensed use limited to: Hong Kong Polytechnic University. Downloaded on May 27,2024 at 10:28:24 UTC from IEEE Xplore. Restrictions apply.

[35] P. Christodoulou, "Soft actor-critic for discrete action settings," 2019. [Online]. Available: http://arxiv.org/abs/1910.07207

[36] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," in *Proc. Int. Conf. Learn. Representations*, 2016, *arXiv:1506.02438*.

[37] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.

[38] M. Mitzenmacher, "The power of two choices in randomized load balancing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 10, pp. 1094–1104, Oct. 2001.

Hongcai Lin received the B.E. degree from South China University of Technology, China, 2020. He is working toward the master's degree with the School of Software Engineering, South China University of Technology, China. His research interests include edge computing and reinforcement learning.

Lei Yang received the B.Sc. degree from Wuhan University, in 2007, the M.Sc. degree from the Institute of Computing Technology, Chinese Academy of Sciences, in 2010, and the Ph.D. degree from the Department of Computing, Hong Kong Polytechnic University, in 2014. He has been a Visiting Scholar with Technical University Darmstadt, Germany, from November 2012 to March 2013. He is currently an Associate Professor with the School of Software Engineering, South China University of Technology, China. His research interests include cloud and edge computing, networking and distributed computing, and Internet of things with special focus on task scheduling and resource management. He has published more than 50 papers in major conferences such as IEEE INFOCOM, PERCOM and top journals such as IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON SERVICES COMPUTING, and *TKDD*. He is a Program Committee Member for many international conferences.

Hao Guo received the master's degree in software engineering from Jilin University, in 2021. He is working toward the Ph.D. degree with the School of Software Engineering, South China University of Technology, China. His research interests include distributed machine learning systems, edge computing, and deep reinforcement learning.

Jiannong Cao (Fellow, IEEE) received the B.Sc. degree in computer science from Nanjing University, China, in 1982, and the M.Sc. and Ph.D. degrees in computer science from Washington State University, USA, in 1986 and 1990, respectively. He is currently the Otto Poon Charitable Foundation Professor in data science and the Chair Professor of distributed and mobile computing with the Department of Computing, Hong Kong Polytechnic University, Hong Kong. He is also the Director of the Internet and Mobile Computing Lab at the Department and the Associate Director of the University Research Facility in big data analytics. His research interests include parallel and distributed computing, wireless networks and mobile computing, big data and cloud computing, pervasive computing, and fault tolerant computing. He has co-authored five books in Mobile Computing and Wireless Sensor Networks, co-edited nine books, and published over 600 papers in major international journals and conference proceedings. He is a Distinguished Member of ACM and a Senior Member of China Computer Federation (CCF).

Authorized licensed use limited to: Hong Kong Polytechnic University. Downloaded on May 27,2024 at 10:28:24 UTC from IEEE Xplore. Restrictions apply.