Dynamic Task Offloading in Edge Computing Based on Dependency-Aware Reinforcement Learning

Xiangchun Chen, Jiannong Cao, *Fellow, IEEE*, Yuvraj Sahni, *Member, IEEE*, Shan Jiang, and Zhixuan Liang

***Abstract*—Collaborative edge computing (CEC) is an emerging computing paradigm in which edge nodes collaborate to perform tasks from end devices. Task offloading decides when and at which edge node tasks are executed. Most existing studies assume task profiles and network conditions are known in advance, which can hardly adapt to dynamic real-world computation environments. Some learning-based methods use online task offloading without considering task dependency and network flow scheduling, leading to underutilized resources and flow congestion. We study Online Dependent Task Offloading (ODTO) in CEC, jointly optimizing network flow scheduling to optimize quality of service by reducing task completion time and energy consumption. The challenge of ODTO lies in how to offload dependent tasks and schedule network flows in dynamic networks. We model ODTO as the Markov Decision Process (MDP) and propose an Asynchronous Deep Progressive Reinforcement Learning (ADPRL) approach that optimize offloading and bandwidth decisions. We design a novel dependency-aware reward mechanism to address task dependency and dynamic network. Extensive experiments on the Alibaba cluster trace dataset and synthetic dataset indicate that our algorithm outperforms heuristic and learning-based methods in average task completion time and energy consumption.**

***Index Terms*—Collaborative edge computing, deep reinforcement learning, network flow scheduling, task offloading.**

I. Introduction

THE rapid proliferation of the Internet of Things (IoT) devices with improved computation and communication capacities has spurred numerous innovative applications, such as autonomous vehicles, smart healthcare, and metaverse [1], [2]. Complex services offered by these applications, such as immersive 3D world rendering in the metaverse [2], are always data-driven and computation-intensive tasks, which involve many dependent components and a large amount of data carried by components. These services impose critical requirements in low latency, which can hardly be provided by centralized cloud computing. Collaborative edge computing (CEC) has been introduced to support such services and applications. CEC has emerged as a distributed computing paradigm where edge nodes collaborate by sharing data and computational resources to achieve individual and global objectives [3], [4].

Task offloading is one of the key enabling technologies in CEC, which refers to the transmission of resource-intensive computational tasks from end devices to a resource-rich platform (i.e., edge nodes). In task offloading, we decide when and at which edge node each task is executed, considering the task profiles, available resources, network conditions, etc. Data transfer time significantly influences task completion time, especially in dependent tasks, because dependent tasks are data-intensive and require access to data distributed throughout the network. However, existing works [5], [6], [7], [8] fail to adequately address the challenge of integrated network flow scheduling, which can result in flow congestion and inefficient performance.

We study the problem of online dependent task offloading (ODTO) in CEC, jointly considering network flow scheduling. The objective is to optimize Quality of Service (QoS) by reducing task completion time and energy consumption. Our research addresses several critical challenges, including: 1) How to simultaneously offload tasks and schedule bandwidth in an online setting and 2) How to optimize the task schedule considering task dependencies.

To address these challenges, we propose an Asynchronous Deep Progressive Reinforcement Learning (ADPRL) algorithm. It is empowered by a Deep Deterministic Policy Gradient (DDPG) strategy with asynchronous data collection and a dependency-aware reward mechanism. We model ODTO as the Markov Decision Process (MDP). Through a dependency-aware reward mechanism, we progressively adjust the offloading decisions by taking two important factors into account: the estimated completion time of the selected tasks and the dependency of the selected tasks in the whole DAG. Through asynchronous data collection, we accelerate the model training.

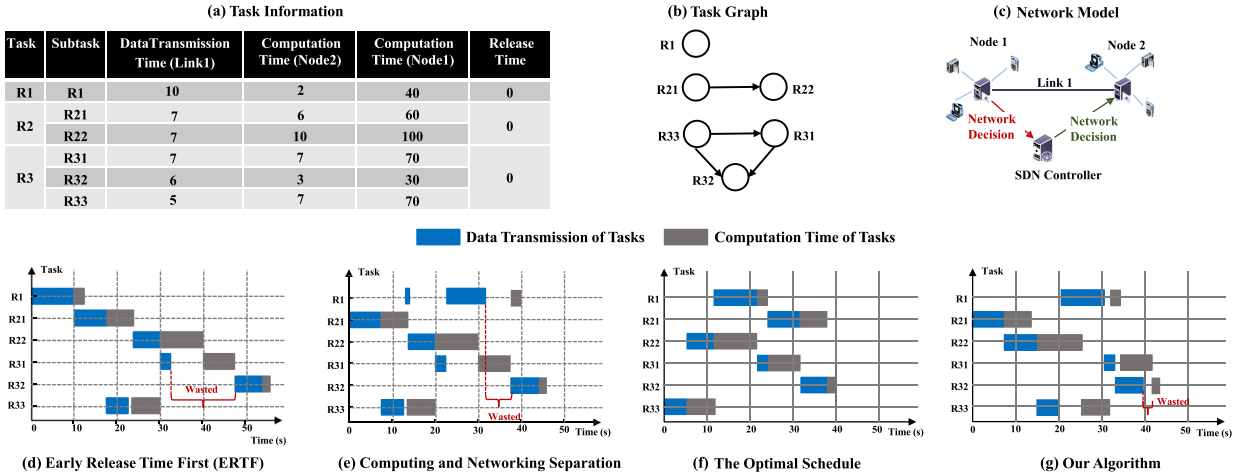
The novelty of this work lies in leveraging the dependency-aware reward mechanism to address task dependency issues and balancing the tradeoff between task completion time and energy consumption. By employing the dependency-aware reward mechanism and reinforcement learning, we can effectively respond to the evolving demands of tasks, resource needs, and

Manuscript received 9 March 2023; revised 3 March 2024; accepted 11 March 2024. Date of publication 27 March 2024; date of current version 7 June 2024. This work was supported by in part the Research Institute for Artificial Intelligence of Things, in part by the Hong Kong Polytechnic University, in part by the Hong Kong (HK) Research Grant Council (RGC) General Research Fund (GRF) under Grant PolyU 15220922, and in part by the Innovation and Technology Fund - Mainland-Hong Kong Joint Funding Scheme (ITF-MHKJFS) under Grant MHP/013/21. Recommended for acceptance by J. Ren. *(Corresponding author: Shan Jiang.)*

Xiangchun Chen, Jiannong Cao, Shan Jiang, and Zhixuan Liang are with the Department of Computing, Hong Kong Polytechnic University, Hong Kong, SAR, China (e-mail: csxcchen@comp.polyu.edu.hk; csjcao@comp.polyu.edu.hk; cs-shan.jiang@polyu.edu.hk; zhixuan.liang@connect.polyu.hk).

Yuvraj Sahni is with the Department of Building Environment and Energy Engineering, Hong Kong Polytechnic University, Hong Kong, SAR, China (e-mail: yuvraj-comp.sahni@polyu.edu.hk).

Digital Object Identifier 10.1109/TCC.2024.3381646



Task	Subtask	DataTransmission Time (Link1)	Computation Time (Node2)	Computation Time (Node1)	Release Time
R1	R1	10	2	40	0
R2	R21	7	6	60	0
	R22	7	10	100	
R3	R31	7	7	70	0
	R32	6	3	30	
	R33	5	7	70	

Fig. 1. Scheduling results of different methods for the tasks in CEC system architecture.

network conditions in real-time. Our reward function enables the deep reinforcement learning model to adaptively explore the tradeoff between task completion time and energy consumption. This method leads to more informed decisions about task dependency and dynamic network and more efficient task offloading and execution. The major contributions of this work are summarized as follows:

- We explore task dependency and dynamic network in online joint task offloading and bandwidth allocation in CEC. We investigate a novel optimization problem for efficient online dependent task offloading: ODTO, which is NP-hard.
- We build an optimization framework for online joint task offloading and bandwidth allocation in CEC. The objective is to optimize QoS by reducing task completion time and energy consumption of tasks. We thoroughly investigate the tradeoff between task completion time and energy consumption.
- We propose ADPRL, an asynchronous deep progressive reinforcement learning algorithm, to optimize task offloading, bandwidth allocation, and waiting time decisions. To address task dependency and dynamic network issues, we design a novel dependency-aware reward mechanism.
- We conduct extensive simulation experiments using the Alibaba cluster trace dataset and synthetic dataset, covering a wide range of network topologies, task numbers, and device numbers that correspond to the characteristics of real-world applications and computation environments. The results show the superiority of our method compared to both heuristic and learning-based algorithms.

II. A Motivating Example

In a simple case shown in Fig. 1, tasks ($R1$, $R2$, $R3$) originate from an edge node (Node 1) and are dispatched to another edge node (Node 2). In Fig. 1(a)), we provide detailed task and subtask information, including data transmission time of links (Link 1), computation time of nodes (Node 1 and Node 2), and the specific release time for each task and subtask. In Fig. 1(b)), the task graph illustrates the dependencies among tasks ($R1$, $R2$, and $R3$). Fig. 1(c)) presents a network model, elucidating the communication framework.

As shown in Fig. 1(f)), the completion time of the optimal schedule is 40 units. As depicted in Fig. 1(f)), the optimal task schedule is the sequence of $R21 \rightarrow R33 \rightarrow R22 \rightarrow R1 \rightarrow R31 \rightarrow R32$, achieving a completion time of 40 units. As shown in Fig. 1(d)), a simple heuristic method is suboptimal. We adopt a widely used heuristic task scheduling algorithm, Earliest Release Time First (ERTF). The inefficiency of ERTF arises from scheduling tasks solely based on release time, neglecting dependencies, and overall task execution and data communication needs. For example, critical tasks with later release time may be scheduled later than less critical ones, leading to longer completion time and inefficient resource utilization. Fig. 1(d)) shows that according to ERTF, the scheduling sequence is $R1 \rightarrow R21 \rightarrow R22 \rightarrow R31 \rightarrow R32 \rightarrow R33$ and the completion time is more than 50 units, while the optimal schedule can finish all tasks in 40 units (Fig. 1(f)). As shown in Fig. 1(e), scheduling computing and networking separately without dependency information is also suboptimal. Because it overlooks the interdependence between task execution and data communication. This can lead to suboptimal scheduling, where tasks ready for execution are delayed due to data unavailability or underutilization of network resources. It can result in increased completion time and inefficient resource utilization. Set the completion time as the task execution time plus flow transmission time. In this case, task preemption is allowed, the processing sequence is $R21 \rightarrow R33 \rightarrow R22 \rightarrow R31 \rightarrow R11 \rightarrow R32$, leading to a completion time exceeding 40 (Fig. 1(e)).

Therefore, it is evident that existing scheduling methods cannot optimally handle task dependencies. It is crucial to consider dependency information in task scheduling, and the joint scheduling of task offloading and network flow is necessary for achieving optimal performance.

TABLE I
MAIN NOTATIONS USED IN THE PAPER

Symbol	Definition
$R_{i,j}$	Release time of subtask j in task i
ID_i	Amount of input data required for task i
$CL_{i,j}$	Computation load associated with each subtask j in task i
$D_{i,j,k}$	Data transmission volume between dependent subtask j and k in task i
$Pd_{i,j}$	Predecessor subtasks for subtask j in task i
CT_i	A task in the set CT
$PS_{n,t}$	Average processing speed of an edge node n at time t
$\delta_{i,j,n,t}$	Binary decision variable for task offloading at edge node n
$PT_{n,i,j}$	Computation time at edge node n for subtask j in task i
ξ_f	Begin time of flow f transmission
$Ge_{f,i,j}$	Binary variable indicating if subtask j in task i generates flow f
$W_{n,i,j}$	Waiting time of subtask j in task i at edge node n
$FT_{i,j}$	Finish time of subtask j in task i
TT_i	Finish time of task i
$\eta^t_{f,i,j}$	Bandwidth allocated to the flow f for subtask j in task i at time t
$\nu_{f,i,j}$	Maximum path bandwidth available to flow f for subtask j in task i
$BT_{f,i,j}$	Flow communication time of flow f for subtask j in task i
ϵ_f	Waiting time of flow f at the edge node
WB_f	Waiting time of flow f for links on the path
EC_i	Total energy consumption of task i
P_{Tx}	Transmission power level
P_{Rx}	Reception power level
UT_i	Uplink transmission time for task i
DT_i	Downlink transmission time for task i
$EC^c_{i,j}$	Energy consumption of subtask j in task i during computation
P^c_n	Computational power of edge node n
$EC^x_{i,j}$	Energy consumption of subtask j in task i during transmission
J	Quality of service metric
λ_t	Scalar weight for task completion time in QoS
λ_e	Scalar weight for energy consumption in QoS
EC	Overall energy consumption for an offloading plan
P^x_u	Transmitting power of edge link u.

III. RELATED WORK

A. Offline Task Offloading in Edge Computing

In the realm of edge computing, the scheduling and offloading of tasks have been studied extensively. A variety of approaches [9], [10], [11], [12] have been explored. These approaches range from algorithmic solutions for computation offloading and bandwidth allocation to heuristic methods for task offloading and resource optimization. However, these methods often overlook task dependencies, impacting application service quality and resource utilization. Additionally, the scheduling of dependent tasks has been investigated [13], [14], [15]. Techniques such as the heterogeneous earliest finish time algorithm by Topcuoglu et al. [13], Tetrium by Hung et al. [14] for geo-distributed clusters, and Sundar et al.'s [16] heuristic algorithm for cloud computing systems have been proposed. Despite their effectiveness, these methods primarily focus on offline scenarios and face challenges in adapting to online environments due to the unpredictability of task profiles.

B. Online Task Offloading in Edge Computing

In edge computing, online task offloading has garnered significant attention. Initial efforts [17], [18] focused on heuristic scheduling methods. However, the effectiveness of these heuristics is limited by the unpredictability of task arrivals and dynamic network conditions. Deep Reinforcement Learning (DRL) has emerged as a promising approach to address these challenges. Chen et al. [19] explored deep Q-Learning and double deep Q-networks to optimize offloading strategies and reduce long-term delays. However, these methods often overlook complex scenarios like multi-hop offloading and high-dimensional action spaces. Recent advancements have addressed these limitations. Techniques ranging from policy gradient DRL [5], [20], [21], meta reinforcement learning [6], Long Short-Term Memory (LSTM)-enhanced Deep Q Network (DQN) [7], to actor-critic algorithms [8], [22] have been employed for various aspects of task offloading, including latency reduction, energy efficiency, and handling task dependencies. Despite these advancements, a common limitation is the lack of integrated network flow scheduling, which can result in network congestion and sub-optimal completion time.

IV. SYSTEM MODEL AND PROBLEM FORMULATION

A. System Model

The CEC system operates in a time-slot manner. The timeline is divided into K (K is a positive integer) time frames (e.g., second, milliseconds). Each time frame can be regarded as the composition of time slots with length T, where T is also a positive integer. Given a sequence of time slots $\{0, 1, \ldots, K * T - 1\}$, we define t as the beginning of each time frame $[t, t + T - 1]$, where $t = k * T$ and k represents a non-negative integer. Fig. 2(a) shows the architecture of a collaborative edge computing system where the intelligence is distributed and pushed within the network by sharing computation resources and data between the network of edge nodes. The system architecture includes a software-defined networking (SDN) controller and edge nodes connected to each other. The role of the SDN controller with global knowledge is responsible for making the decision for offloading tasks and scheduling bandwidth in the network. It includes different functional components responsible for collecting information and making scheduling decisions. Edge nodes can be heterogeneous in computation capacity.

Definition (Path): A *path* denotes a sequence of connected nodes and links through which data can be routed from its source to its destination. Each node represents a device (such as routers, edge devices or end devices), and each link represents the connection between these nodes.

Definition (Flow): A *flow* is defined as a sequence of packets passing an observation point of a path during a certain time interval. All packets that are part of a specific flow share a set of common properties, which may include, but are not limited to, packet headers, source and destination addresses, protocols used, and quality of service parameters.

Task Model is defined by a collection of DAG tasks, denoted as CT. This collection CT includes each task CT_i, denoted as $CT = \{CT_i | 1 \leq i \leq \beta\}$, where β is the total number of tasks. Each task i is modelled as a DAG $CT_i = (T_i, P_i)$. T_i is the set of dependent subtasks in task i. $T_i = \{j | 1 \leq j \leq N_i\}$, where N_i is the number of subtasks in task i. $P_i = \{D_{i,j,k} | j \in Pd_{i,k}, j \in$

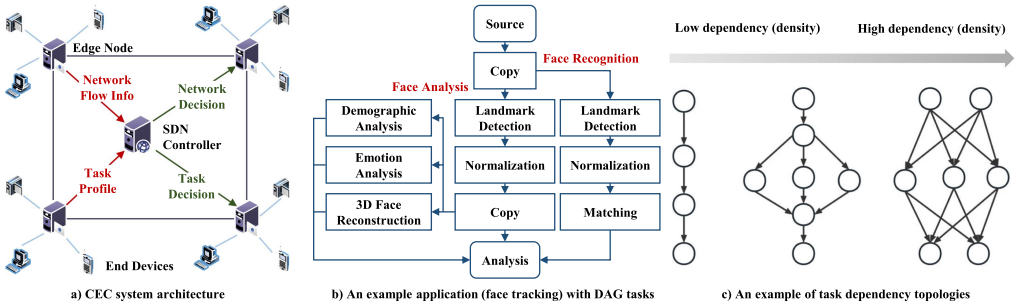


Fig. 2. System architecture of collaborative edge computing and example application with DAG tasks.

$T_i, k \in T_i\}$ is the set of dependencies (weights) among the subtasks in task i. Each DAG can have general dependencies. Task dependency is the degree to which the data and execution of different subtasks within a composite task rely on one another's outputs or processing states. It reflects the interconnectedness and the reliance of subtasks on the sequential or parallel flow of information [23]. Fig. 2(c) shows an example of general task dependencies by DAG topologies, ranging from low to high task dependency, depicted by the density of connections between subtasks. The amount of input data required for task i is ID_i. Each subtask j in task i is associated with the computation load $CL_{i,j}$. The weight of the link connecting subtask j and k of task i is $D_{i,j,k}$, which represents the data transmission volume between dependent subtask j and k in task i. The predecessors for subtask j in task i is $Pd_{i,j}$.

Network Model is defined as a connected graph $G = (V, E)$. Inside, $V = \{k|1 \leq k \leq \alpha\}$ is the set of edge nodes, where α is the total number of edge nodes. E is the set of edge links, $E = \{e_{j,k,w}|1 \leq j \leq p, c_{k,w} = 1\}$, where p is the number of edge links. $c_{k,w}$ indicates whether edge node k and w are connected, with a value of "1" representing the connection. Otherwise, it is set to "0". $PS_{n,t}$ indicates the average processing speed of an edge node n at time t. Table I presents the main notations used in the paper.

B. Problem Formulation

Computation offloading management contains a series of operations to handle task computation demands in the network area. It includes the task offloading decision and bandwidth allocation decision of links. In a stochastic environment, the computation demands of tasks on edge nodes are uncertain and subject to constant change. Fig. 3 illustrates the dynamic task offloading framework. The offloading scheduler will make joint decisions of task offloading, bandwidth allocation, and waiting time at each time slot.

1) Task Offloading: Considering the device heterogeneity, tasks can be offloaded to an edge device at a multi-hop distance. Therefore, the CEC system needs to decide which edge node the tasks will be offloaded. Task offloading decisions are significantly influenced by task dependencies, as these dependencies dictate the order and timing of task execution.

Definition (Task Offloading Decision): Let $\delta_{i,j,n,t} \in \{0, 1\}$ be a binary decision variable to indicate whether the computing

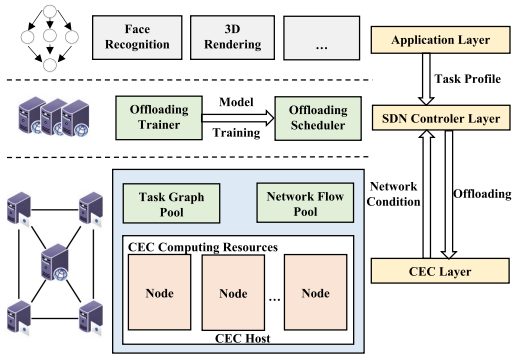


Fig. 3. Dynamic task offloading framework.

power at edge node n is allocated to subtask j in task i at time t. The task dependencies imply that the value of $\delta_{i,j,n,t}$ may be contingent on the completion of predecessor tasks, thereby affecting the offloading decision.

Definition (Task Computation Time): Let $PT_{n,i,j}$ represent the computation time of edge node n for subtask j in task i. It is composed of four parts: a) the time when the task is uploaded to the edge nodes, b) the waiting time of the task at an edge node, c) the calculation time of the task at the edge nodes, and d) the download time when the task result is returned to the user. The waiting time of subtask j in task i on node n $W_{n,i,j}$ becomes particularly crucial, as it must account for the completion of all preceding dependent tasks before the current task can commence processing. This directly impacts the computation time $PT_{n,i,j}$ and, in turn, influences the offloading decision. Computation time $PT_{n,i,j}$ is defined as:

$$PT_{n,i,j} = CL_{i,j}/(PS_{n,t}) + W_{n,i,j} \quad if \quad \delta_{i,j,n} = 1 \tag{1}$$

$ST_{i,j}$ is the start time of subtask j in task i. ξ_f is the begin time of flow f transmission. $Ge_{f,i,j} = 1 \in \{0, 1\}$ is a binary variable, which has the value "1" if subtask j in task i generates flow f. Otherwise, it is set to "0". ST_i is the start time of task i. $R_{i,j}$ is the release time of subtask j in task i.

2) Bandwidth Allocation: Task dependencies affect bandwidth allocation decisions. The allocation of bandwidth to specific flows for subtasks must consider the immediate demands of the subtask and the requirements of dependent tasks that follow.

Definition (Bandwidth Allocation Decision): Let $\eta^t_{f,i,j} \in [0, \nu_{f,i,j}]$ be a continuous decision variable to indicate the bandwidth allocated to the flow f for subtask j in task i at time t. $\nu_{f,i,j}$ represents the bandwidth maximum available to flow f for subtask j in task i on its path, determined by the minimum bandwidth of all links in the path. The task dependency determines the priority and amount of bandwidth allocated, as dependent tasks might require immediate and substantial bandwidth to ensure efficient data transfer and timely task completion.

$BW_{u,t}$ is the remaining bandwidth of edge link u at time t. χ_f is the finish time of flow f. $\omega_{f,i,j}$ is a binary variable, which has the value "1" if subtask j in task i requires flow f. Otherwise, it is "0". Assume that the routing path is known, the remaining bandwidth maximum $\nu_{f,i,j}$ is calculated as:

$$\nu_{f,i,j} = \min_{j \in path} BW_{u,t} \tag{2}$$

Definition (Flow Communication Time): Let $BT_{f,i,j}$ represent the flow communication time of flow f for subtask j in task i. ϵ_f is the waiting time of the flow f at the edge node. WB_f is the waiting time of flow f for links on the path. σ_f is the size of flow f. A high flow waiting time of flow f WB_f value may suggest reevaluating and possibly increasing bandwidth allocation to accommodate dependent tasks' requirements. This directly affects the flow communication time $BT_{f,i,j}$ and consequently influences the bandwidth allocation decision.

The flow transmission time $BT_{f,i,j}$ of flow f for subtask j in task i is now recalculated as:

$$BT_{f,i,j} = \max_{k \in Pd_{i,j}} FT_{k,j} + \frac{ID_{i,j}}{\eta^t_{f,i,j}} + \epsilon_f \tag{3}$$

where ϵ_f represents the waiting time of the flow f at the edge node. The finish time $FT_{i,j}$ of subtask j in task i is calculated as:

$$FT_{i,j} = R_{i,j} + PT_{n,i,j} + BT_{f,i,j} \tag{4}$$

In edge computing, the upload latency for task i to an edge node k is denoted as UT_i. During upload, task i transmits a substantial data volume to node k via the shortest network path. Let PR_i be the data amount of the task processing result. The transmission bandwidth for uploading task UB_i and downloading results DB_i is constrained by the lowest bandwidth link on the path. The uploading time UT_i and downloading time DT_i for task i is calculated as $\frac{ID_i}{UT_i}$ and $\frac{PR_i}{DB_i}$, respectively.

The finish time TT_i of task i is calculated as:

$$TT_i = UT_i + \max_{j \in CT_i} FT_{i,j} + DT_i \tag{5}$$

3) Energy Consumption: Energy consumption is a critical factor that impacts system performance in CEC, which encompasses both computation and transmission energy costs.

Computation Energy Consumption: Energy consumption during local task execution is dominated by computational processes. The energy consumption $EC^c_{i,j}$ of subtask j in task i is formulated as:

$$EC^c_{i,j} = P^c_n * PT_{i,j} \tag{6}$$

Where P^c_n represents the computational power of the edge node n while c is a power coefficient [20]. $PT_{i,j}$ is the computation time of subtask j in task i.

Transmission Energy Consumption: When a task is offloaded, transmission energy consumption $EC^x_{i,j}$ of subtask j in task i is primarily due to data transmission. P^x_u denotes the transmitting power of edge link u.

$$EC^x_{i,j} = P^x_u * BT_{f,i,j} \tag{7}$$

The energy consumption EC_i of task i is calculated as:

$$EC_i = P_{Tx} * UT_i + \sum_{j \in CT_i} \left(EC^c_{i,j} + EC^x_{i,j}\right) + P_{Rx} * DT_i \tag{8}$$

Where P_{Tx} and P_{Rx} are the transmission and reception power levels, respectively, and UT_i and DT_i are the uplink and downlink transmission time for task i.

The overall energy consumption for an offloading plan of task i, considering both latency and energy, is given by:

$$TT = \sum_{1 \le i \le \beta} TT_i/\beta \tag{9}$$

$$EC = \sum_{1 \le i \le \beta} EC_i \tag{10}$$

Quality of service (QoS) serves as a metric for evaluating the effectiveness of an offloading plan, taking into account both task completion time and energy consumption. As proposed in [20], the QoS can be quantified as a weighted sum where task completion time TT and energy consumption EC are normalized against their respective maximum values, $TT_{\max}$ and $EC_{\max}$.

$$J = -\left(\lambda_t * \frac{TT}{TT_{\max}} + \lambda_e * \frac{EC}{EC_{\max}}\right) \tag{11}$$

where λ_t and $\lambda_e \in [0, 1]$ are scalar weights of task completion time and energy consumption respectively.

Guidelines for Configuring Weighting Factors: We present a comprehensive framework for the adjustment of weighting factors, λ_t and λ_e, tailored to specific real-world applications. The framework initiates with normalization, where ACT and EC are normalized to ensure they are on a similar scale by dividing each by their maximum observed or possible value within the system. Weight assignment then allocates weights λ_t and λ_e, based on the importance of latency and energy consumption in real-world applications, ensuring $\lambda_t + \lambda_e = 1$. In the calculation step, these normalized values of ACT and EC are used to compute QoS, quantifying QoS as the weighted average of latency and energy differences. Finally, in adjustment and testing, we adjust the weights based on real-world testing and feedback, recognizing that different applications might require different balances between latency and energy consumption. We detail empirical values for λ_t and λ_e for three types of real world applications.

1) *Latency-Prioritized Applications:* In applications where latency is critical, such as those involving real-time decision-making like pedestrian reidentification and traffic monitoring, a higher weight is assigned toλ_t (> 0.5)

to prioritize latency reduction, while λ_e is set lower (< 0.5). An example can be seen in [24], which minimizes latency in real-time video surveillance for pedestrian reidentification.

2) *Balanced-Priority Applications:* Applications requiring a balanced configuration to latency and energy consumption, such as autonomous driving systems. Both λ_t and λ_e are crucial, with a slight preference for λ_t to ensure system responsiveness while maintaining energy efficiency. An example of autonomous driving services can be seen in [25], which balancing latency and energy consumption.
3) *Energy-Prioritized Applications:* For scenarios where energy consumption is a primary concern, such as remote environmental monitoring systems, prioritizing $\lambda_e > 0.5$ and $\lambda_t < 0.5$ emphasizes energy efficiency over immediate task completion. This configuration is particularly relevant for battery-powered devices deployed in isolated locations, where extending operational life is crucial. An example is detailed in [26], where a comprehensive environmental monitoring framework integrates energy-aware optimization components.

C. NP-Hardness Proofs

Theorem 1: The online joint dependent task offloading and bandwidth allocation problem in $G = (V, E)$ is NP-hard.

Proof: To establish the NP-hardness of the online joint dependent task offloading and bandwidth allocation problem in $G = (V, E)$, we approach by demonstrating that even a simplified version of this problem can be shown to be NP-hard by reduction from a known NP-hard problem, specifically the Multi-dimensional Knapsack Problem (MKP) [27].

The MKP involves selecting a subset of N items to maximize the total value without exceeding the multi-dimensional capacity constraints of a knapsack. Each item n has a value $v(n)$ and consumes a certain amount of resource $r_d(n)$ in dimension d, with the knapsack having a capacity $C(d)$ in each dimension d. The objective is to maximize the total value of the selected items while respecting the capacity constraints in all dimensions.

To relate this to our problem, we consider each task or subtask in the online joint dependent task offloading and bandwidth allocation problem as an item in MKP, where the "value" of an item corresponds to the negative of the offloading and execution cost of a subtask on an edge node. The multi-dimensional resources in MKP correspond to the computation and bandwidth resources in our problem, and the capacities in MKP are analogous to the computation and bandwidth capacities of the edge nodes.

By demonstrating that any instance of MKP can be mapped to our problem, we establish its NP-hardness. The online nature, requiring decisions without future knowledge, further maintaining NP-hardness. Thus, the online joint dependent task offloading and bandwidth allocation problem in $G = (V, E)$ is NP-hard.

D. ILP Formulation of the Offline Problem Version

We formulate an integer linear programming (ILP) formulation for the joint dependent task offloading and bandwidth allocation problem. For the offline version, the objective of the problem is to optimize the QoS by reducing the average completion time and energy consumption of all tasks. The joint dependent task offloading and bandwidth allocation problem can be formatted as:

$$\max J \tag{12}$$

This is equivalent to:

$$\min \lambda_t * \frac{TT}{TT_{\max}} + \lambda_e * \frac{EC}{EC_{\max}} \tag{13}$$

Subjects to Constraints:

$$FT_{i,j} \leq ST_{i,k} \quad \forall CT_{i,j} \in Pd_{i,k}, \tag{14}$$

$$R_{i,j} \leq \xi_f \quad \text{if} \quad Ge_{f,i,j} = 1, \tag{15}$$

$$\chi_f \leq ST_i \quad \text{if} \quad \omega_{f,i,j} = 1, \tag{16}$$

$$P_n^c \leq P_{\max,n}^c \quad \forall 1 \leq n \leq \alpha, \tag{17}$$

$$P_u^x \leq P_{\max,u}^x \quad \forall 1 \leq u \leq p \tag{18}$$

- Constraint (14) enforces that a subtask can start only after preceding subtasks have finished.
- Constraint (15) enforces that the network flow starts only after the preceding subtask.
- Constraint (16) enforces that the start time of a task is at least equal to the finish time of the required flow.
- Constraint (17) ensures that computation power P_n^c of edge node n does not exceed its maximum power threshold $P_{\max,n}^c$, preventing node overloading.
- Constraint (18) ensures that the transmission power P_u^x of edge link u does not exceed its maximum power threshold $P_{\max,u}^x$, managing the power used in wireless communication.

The ILP formulation can provide an optimal solution, but it assumes request arrivals for all time slots are known. This makes it impractical for real-world scenarios and limits its applicability to dynamic problems.

V. Asynchronous Deep Progressive Reinforcement Learning for Online Dependent Task Offloading

This section models the ODTO problem as the Markov Decision Process and presents our asynchronous deep progressive reinforcement learning algorithm.

A. Markov Decision Process

We employ the Markov Decision Process to model the ODTO problem. Here, scheduling actions involve selecting an offloading target and bandwidth allocation for each incoming task, initiating a state change described by $Pr[S_{t+1} = s' | S_t = s, A_t = a]$. In this expression, S_t and S_{t+1} represent the current and subsequent states, respectively, while A_t denotes the scheduler's action. The scheduler, acting as the agent, aims to identify actions that optimize state transitions towards maximizing overall rewards, effectively addressing the task offloading challenge.

B. State Space

When scheduling the task CT_i, the state of the CEC system depends on the scheduling results of the previous tasks and network flows. Hence, we define the state space as a combination of the task information G (including DAG topologies and task profiles) and the network information E (including bandwidth conditions of links and load conditions of edge nodes). Let $A_{1:i}$ denotes the offloading plan for the sequence of tasks from 1 to i. The state S is represented by

$$S = \{s(t)|s(t) = (G, E, A_{1:i})\} \tag{19}$$

$$G = \{g_i|g_i = (CT_i, T_i, P_i)\} \tag{20}$$

$$E = \{Te_1(t), \ldots Te_p(t), Td_1(t), \ldots, Td_\alpha(t)\} \tag{21}$$

Specifically, task information G comprises of three elements: 1) an index CT_i of the task; 2) a vector T_i of dependent subtasks in task CT_i; 3) a vector P_i of dependencies (weights) among subtasks in task CTi. Network information E is the condition of CEC network environment, consisting of: 1) the bandwidth conditions of links represented by the set of the waiting time for available bandwidth for edge links $\{Te_1(t), \ldots, Te_j(t), \ldots, Te_p(t)\}$ in which $Te_j(t)$ represents the waiting time for available bandwidth for edge link j; 2) the load conditions of edge nodes represented by the set of waiting time for available computation power for edge nodes $\{Td_1(t), \ldots, Td_j(t), \ldots, Td_\alpha(t)\}$ in which $Td_n(t)$ represents the waiting time for available computation power for edge nodes n at time slot t.

C. Action Space

By observing the current environment state, the agent will take actions accordingly. The key step of task scheduling is to choose the offloading decision that indicates which device assigned to the current task and the amount of bandwidth of the according flow. The actions at time t are defined as:

$$a_t = \{a_{i,t}|a_{i,t} = \{q_{i,t}, \eta_{i,t}, \epsilon_{i,t}\}\} \tag{22}$$

Specifically, the action $a_{i,t}$ at time t for task i has three vectors, the first vector $q_{i,t} = \{\delta_{i,1,n,t}, ..., \delta_{i,j,n,t}, ..., \delta_{i,\alpha,n,t}\}$ where α is the number of edge nodes and $\delta_{i,j,n,t}$ indicates whether subtask j in task i is allocated to edge node n. The second vector $\eta_{i,t} = \{\eta_{i,1}, ..., \eta_{i,f}, ..., \eta_{i,f_n}\}$ where f_n is the number of flows and $\eta_f \in [0, \nu_{f,i,j}]$ represents the bandwidth allocated to flow f. The third vector $\epsilon_{i,t} = \{\epsilon_{i,1}, ..., \epsilon_{i,f}, ..., \epsilon_{i,f_n}\}$ where f_n is the number of flows and $\epsilon_f \in [0, +\infty]$ represents the waiting time of the flow f.

D. Reward Function Design

The reward function denotes the immediate benefit the agent receives after executing an action. We define the reward r_t as the negative value of the weighted sum where task completion time TT and energy consumption EC are normalized against their respective maximum values, $TT_{\max}$ and $EC_{\max}$.

$$r_t = \sum_i - \left(\lambda_t * \frac{TT}{TT_{\max}} + \lambda_e * \frac{EC}{EC_{\max}}\right) \tag{23}$$

Algorithm 1: ADPRL Algorithm.

Input: Training episode length Y, training sample length T; DAG task number β, edge node number α, edge link number p;

1: Randomly initialize actor network A_θ and critic network C_w with weights θ and w, respectively
2: Initialize target network A_θ and C_w with weights w' and θ', respectively
3: $w' \leftarrow w$, $\theta' \leftarrow \theta$
4: Initialize replay memory B
5: **for** episode $\leftarrow 0$ **to** Y **do**
6: Get environment state s_t
7: Initialize a random process N for action exploration
8: **for** $t \leftarrow 1$ **to** T **do**
9: Select action $a_t = (q_t, \eta_t, \epsilon_t)$ according to the current policy and exploration noise
10: In collaborative edge computing, execute action a_t and obtain the reward r_t and new state s_{t+1}
11: Store state transition (s_t, a_t, r_t, s_{t+1}) in pool B
12: Randomly select a minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from B.
13: $y_i \leftarrow r_i + \gamma Q'(s_{i+1}, \mu'_{\theta'}(s_{i+1}); w')$
14: Update the critic network by minimizing the loss $L = 1/N \sum_i (y_i - Q(s_i, a_i; w))^2$
15: Update the actor network as follows: $\nabla_{\theta_\mu} J \approx 1/N \sum_i Q(s_t, a_t; w)|_{s_t=s_i, a=\mu_\theta(s_i)} \nabla_{\theta_\mu} \mu_\theta(s_t)|_{s_t=s_i}$
16: $\theta'_{\mu'} \leftarrow t\theta_\mu + (1-t)\theta'_{\mu'}$ ▷ Update one of the target networks
17: $w'_{Q'} \leftarrow t w_Q + (1-t) w'_{Q'}$ ▷ Update the other target network
18: **end for**
19: **end for**

E. Deep Deterministic Policy Gradient

To better adapt to the CEC network environment, we formulate it as a MDP. The system state transition obtained at time slot t is denoted as (s_t, a_t, r_t, s_{t+1}), which includes state, action, reward and next state. The transition function $m(s'|s, a)$ of the MDP represents the probability of transit from current state s to state s', which can be defined as:

$$m\left(s'|s, a\right) = Pr\left\{s_{t+1} = s'|s_t = s; a_t = a\right\} \tag{24}$$

In Fig. 4, the DDPG model has actor network, critic network and experience pool. The actor-network is policy-based while the critic network is value-based. The experience pool can store each system state record (s_t, a_t, r_t, s_{t+1}). The deterministic strategy μ can be expressed as:

$$\mu : s_t \rightarrow a_t \tag{25}$$

After approximating the strategy with parameter θ, the deterministic strategy is defined as:

$$\mu_\theta\left(a_t|s_t\right) = P\left(a_t|s_t; \theta\right) \tag{26}$$

Then, the objective of the agent is to learn the optimal policy to maximize the progressive reward:

$$J(\theta) = \mathbb{E}_{s_t \sim \rho^\mu, a_t \sim \mu}[r] \tag{27}$$

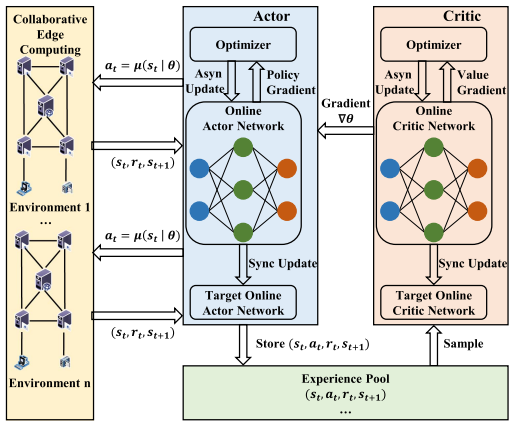


Fig. 4. Diagram of asynchronous deep deterministic policy gradient algorithm.

where ρ^{μ} is the state transition probability given the action distribution μ and r denotes the progressive expected rewards. The term r in the policy gradient estimator leads to high variance, as these returns can vary drastically as the number of episodes increases.

The actor-critic network aims to address this issue by using a function $Q_{\omega}(s, a)$ to approximate the expected returns, and replacing the original return term in the policy gradient estimator. According to the deep determine policy gradient theorem, the gradient of the objective J is:

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{s_t \sim \rho^{\mu}} \left[\nabla_{\theta} \mu_{\theta}(a \mid s_t) \nabla_a Q'(s_t, a)|_{a=\mu_{\theta}(s_t)} \right] \quad (28)$$

The replay buffer stores the prior system state transition experience in a replay memory pool. During training, we randomly sample data (s_t, a_t, r_t, s_{t+1}) from it to increase learning performance. The replay memory pool comprises observed state transitions and gained rewards produced by activities in each time slot.

In order to train the critic network parameters, a batch of stored experience is randomly selected from the replay memory pool as samples. The purpose of the training is to minimize the difference between $Q(s_t, a_t; w)$ and $Q'(s_t, a_t; w')$. To represent the difference between Q and Q', we define a loss function L, which is,

$$L = 1/N \sum_{i} \left(y_i - Q\left(s_t, a_t; w\right) \right)^2 \quad (29)$$

$$y_i = r_i + \gamma Q'\left(s_{i+1}, \mu'_{\theta'}(s_{i+1}); w'\right) \quad (30)$$

where N denotes the number of samples drawn from the experience pool, y_i is the temporal difference (TD) target for Q-learning, and w and w' are parameter of critic network.

The scheduling algorithm is summarized in Algorithm 1. The scheduler first initializes the replay buffer, critic network, and actor network with weight w and θ (lines 1–3 in Algorithm 1). After obtaining the state s_t of the environment, the agent selects the action a_t according to the current policy and exploration noise (lines 5–8 in Algorithm 1). After performing the action and interacting with the environment in the processing pool, the agent will receive the reward r_t and observe the next state s_{t+1} of the environment, then store the state $\langle s_t, a_t, r_t, s_{t+1} \rangle$ into the RB (lines 9-10 in Algorithm 1). Then, the agent randomly samples a mini-batch experience from the B and updates the critic network by minimizing the loss (lines 11–13 in Algorithm 1). After each episode of interaction with the environment, the agent updates the actor network and critic network using the sampled policy gradient and the target network parameters (lines 14–19 in Algorithm 1).

F. Comparison Between ADPRL and Optimal Offline Solution

To further substantiate the efficacy of ADPRL, we present a comparison with the optimal offline solution.

Consider a simple scenario of dynamic edge computing environments depicted in Fig. 1, we compare ADPRL (Fig. 1(g)) and the optimal offline solution (Fig. 1(f)). Here, the completion time is a crucial metric, defined as the aggregate of task execution time and flow transmission time. Importantly, we incorporate task preemption into our scenario to reflect the dynamic nature of real-world edge computing. For the optimal offline solution, the completion time is quantified at 40 units, as depicted in Fig. 1(f). The optimal task schedule is the sequence of $R21 \rightarrow R33 \rightarrow R22 \rightarrow R1 \rightarrow R31 \rightarrow R32$, achieving a completion time of 40 units. Compared to other methods such as Earliest Release Time First (56 units in Fig. 1(d)), and scheduling computing and networking separately (46 units in Fig. 1(e)), our ADPRL algorithm demonstrates a remarkable efficiency(43 units). As depicted in Fig. 1(g), ADPRL schedules tasks in a sequence of $R21 \rightarrow R22 \rightarrow R1 \rightarrow R33 \rightarrow R31 \rightarrow R32$, achieving a completion time of 43 units. This performance is notably closer to the optimal solution, thereby illustrating the effectiveness of our proposed method.

This comparison demonstrates that while ADPRL does not always achieve the theoretical optimum, it significantly outperforms baselines and closely approximates the optimal offline solution, thereby validating the effectiveness of our proposed method in edge computing scenarios.

G. Scalability Analysis on Heterogeneous Edge Nodes

Our algorithm is uniquely equipped to manage the challenges of heterogeneous edge devices. By leveraging the DRL framework, our approach adapts dynamically to the varying computational states and bandwidth conditions of these devices, learning through trial and error. This process is facilitated by a state space that incorporates both computational and bandwidth statuses, enabling the algorithm to optimize offloading decisions about task allocation and bandwidth distribution. These decisions encompass allocating tasks to specific devices and the corresponding bandwidth allocation for the flow. The action space is informed by current device statuses, and a reward function that provides performance feedback. These elements are central to refining the decision-making process. This ensures continual adaptation and optimal alignment with network conditions and device capabilities, enhancing the management of heterogeneous edge environments.

However, there are two main challenges associated with ADPRL in terms of scalability. One challenge is increased signal

transmission delays. As the number of edge nodes increases, the volume of common signals transmitted between the central control and the edge nodes also escalates. This increase in signal traffic results in higher transmission delays, potentially affecting the overall system efficiency. Another challenge is the slower convergence of model training. This is due to the more complex and varied data being processed and the need for more extensive coordination and synchronization across nodes.

While these challenges are crucial to the scalability and performance of CEC systems, our algorithm has certain limitations in directly addressing them. Our work primarily focused on optimizing task offloading, bandwidth allocation, and waiting time decisions under small-scale networks and nodes. In our future studies, we will explore efficient data compression techniques and optimize network protocols to mitigate increased signal transmission delays. For slower model convergence, we will consider advanced parallel training algorithms to accelerate the model training.

VI. Performance Evaluation

A. Simulation Settings

We consider the average completion time (ACT) and energy consumption (EC) as performance metrics. ACT is calculated as the total completion time of tasks divided by the total number of tasks. EC measures the electrical energy used by system components like CPUs and networking equipment [20]. Computation energy consumption is measured by the power used in processing tasks, considering both active and idle states. Transmission energy consumption evaluates the energy used in data transmission, factoring in data volume and network efficiency.

Network Model: We adopt a random topology generator[1] to generate such a network topology. The link weight between two edge nodes represents the link bandwidth. Due to device heterogeneity, the processing speed of edge nodes is chosen from a normal distribution, with a default mean of 40 Mcps (megacycles per second) and a variance of 80%. Similarly, the bandwidth of each link is chosen from a normal distribution, with a default mean of 10 Mbps (megabits per second) and a variance of 80%. These default values may be subject to modifications as needed.

Synthetic Dataset: We adopt a random DAG generator [23] to generate DAG tasks. The properties of each DAG task include task id, subtask id, data size, computation load, release time, and source device. The node and layer number of DAGs are chosen from a normal distribution $[1, 100]$. The number of edges (dependencies) between two layers in the DAG is selected from a uniform distribution $[1, 100]$. The tasks are randomly generated at edge nodes. The data size of each task is generated randomly from a normal distribution with a mean of 500 Mbit and a variance of 80%. The computation load of each task is randomly selected from a normal distribution with a mean of 500 KCC (Kilo Clock Cycles) and a variance of 60%. The release time of each task is randomly selected using a Poisson distribution.

Alibaba cluster trace dataset: Alibaba cluster trace dataset contains the DAG information of real-world workload traces[2]. Due to the lack of certain information in the Alibaba cluster trace dataset, we have modified and used it to generate DAGs. The computation loads of tasks are acquired by properly scaling the value of "start time" minus "end time" multiplied by "plan cpu". The data size of each task is randomly selected using a normal distribution with a mean of $500Mbit$ and variance of 80%. The release time of each task is randomly selected using a Poisson distribution. Tasks are generated at edge nodes randomly. We divide the dataset into two groups, where one group is the training dataset, and the rest is the test dataset.

In our experiments, we set two different objectives: latency optimization (LO) and energy efficiency (EE), following the guidelines for configuring weighting factors outlined in Section IV-B.3. The LO objective focuses on minimizing latency, and therefore we assign $\lambda_t = 1.0$ and $\lambda_e = 0.0$. The EE objective aims to optimize both latency and energy consumption, hence we assign $\lambda_t = \lambda_e = 0.5$. We implement ADPRL on our platform with 32 cores and 2 GPU cards. The training converges at events for around 10,000. The training parameters are described as follows. The replay memory size is 10000. The mini-batch size is 64, representing the number of memories used for each training step. The learning rates of actor and critic networks are relatively 0.001 and 0.002. The reward decay is 0.001. For the setting of asynchronous deep reinforcement learning, the number of servers and workers is 1 and 4, relatively. The training information of each worker will be merged by the server. After training, we acquire two models: one for the LO target and the other for the EE target, which are subsequently deployed to edge nodes. The inference overhead is negligible compared to the time costs of task offloading [20]. We implement four competitive task algorithms of task offloading for performance comparison. The baseline algorithms are as follows.

- *Random:* Tasks are offloaded randomly to the edge nodes, and flows are transmitted randomly through the network links.
- *Local Execution (LE):* Tasks are executed on the edge node where they are generated, negating the need for network flow scheduling.
- *Greedy:* This strategy computes the estimated completion time for each device and prefers the device with the minimal time for offloading. Network flow scheduling follows First Come First Served (FCFS).
- *DQN + FCFS:* Tang et al. [7] proposed a DRL-based online offloading algorithm that incorporates the LSTM, dueling DQN, and double-DQN techniques without considering task dependency.

B. Comparison of ADPRL and Baselines on Testbed

We implement the Testbed with one IP camera, three edge devices, and a master device. For edge devices, we employ one NVIDIA Jetson Xavier NX and two NVIDIA Jetson TX2 units. The Jetson Xavier NX is equipped with a 384-core Volta GPU

[1] [Online]. Available: https://github.com/cesarghali/topology-generator/.

[2] [Online]. Available: https://github.com/alibaba/clusterdata/tree/master/cluster-trace-v2018.

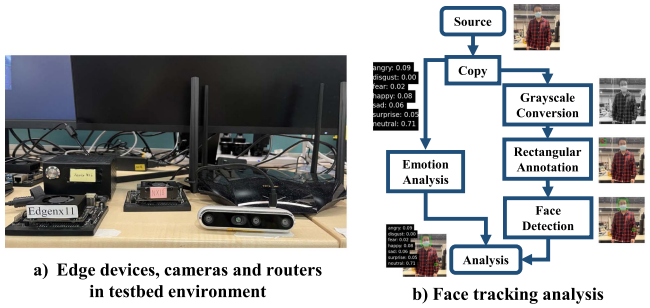


Fig. 5. Demo of pedestrian face tracking analysis. The left corner illustrates the edge devices, cameras and routers in the environment. The captured images and analytics results of a camera are shown on the right.

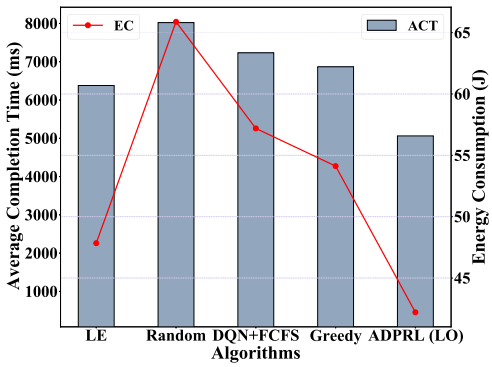


Fig. 6. Comparison of ADPRL (LO) and Baselines on Testbed.

and 8 GB of RAM, whereas each Jetson TX2 features a 256-core GPU along with 8 GB of RAM. Notably, the Jetson Xavier NX boasts significantly higher computational capabilities compared to the TX2 units. In order to simulate a heterogeneous resource environment, the memory of the Jetson TX2 units is artificially limited to 4 GB [24]. The master device is a high-performance workstation, containing four Intel Core i5-8300H processors and 16 GB of RAM, serving as the scheduler. Network connectivity among the edge devices and the camera is established through a router.

Pedestrian Face Tracking Analysis Pipeline: We utilize SSD-MobileNet-V2 [28] and VGG-16 [29] for processing input images, facial localization, and emotion analysis. SSD-MobileNet-V2 is a streamlined deep convolutional network employing depth-wise separable convolutions. VGG-16, with its 16 convolutional layers, excels in extracting complex features from images, a key aspect for our facial feature extraction and emotion analysis tasks. In Fig. 5, we efficiently extract facial features and analyze emotions for effective face tracking. We fine-tune the pre-trained models of SSD-MobileNet-V2 and VGG-16 to optimize them for our specific application. These models are developed under serverless principles and accessed via RESTful APIs [30].

Experiment Result: In Fig. 6, in terms of ACT, ADPRL (LO) exhibits superior performance when compared to the baselines. This can be attributed to the fact that ADPRL (LO)'s sophisticated decision-making process allows for more effective prioritization of tasks, thereby ensuring efficient task completion. In particular, ADPRL (LO) is 36.9% less than Random, 20.6% less than LE, and 26.3% less than Greedy, while also maintaining a 30.0% less ACT compared to DQN+FCFS. In Fig. 6, in terms of EC, ADPRL (LO) outperforms the baselines significantly, further highlighting its efficiency. This efficiency is attributed to its advanced learning mechanisms, which optimize the balance between computational load and energy usage. ADPRL (LO) is 35.9% less than Random, 11.8% less than LE, and 22.0% less than Greedy, while also maintaining a 26.2% less energy consumption compared to DQN+FCFS.

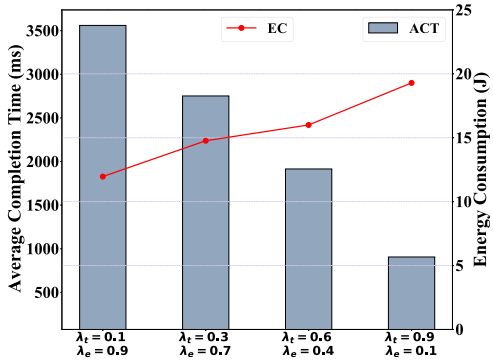


Fig. 7. Tradeoff between task completion time and energy onsumption.

C. Tradeoff Between Task Completion Time and Energy Consumption

We delve into the tradeoff between task completion time and energy consumption, a pivotal aspect of our system's performance. As mentioned earlier, we set λ_t as the weighting factor of task completion time and λ_e as the weighting factor of energy consumption to study the tradeoff between energy consumption and task completion time.

In Fig. 7, we study the performance of ADPRL on the Alibaba cluster trace dataset with different values of $\lambda_t = 0.1, 0.3, 0.6, 0.9$ and $\lambda_e = 0.9, 0.7, 0.4, 0.1$ in four scenarios. In the first two scenarios, we assign lower values to λ_t (0.1 and 0.3) and higher values to λ_e (0.9 and 0.7), emphasizing energy efficiency. In the latter two scenarios, the focus shifts to task completion time by setting higher values for λ_t (0.6 and 0.9) and lower values for λ_e (0.4 and 0.1). When the weighting factor for energy consumption (λ_e) is increased, we observe a reduction in energy consumption (EC), as the system is optimized more towards energy efficiency. With the decrease in λ_t, the system places less emphasis on completion time, potentially adopting strategies that are more energy-efficient but slower.

D. Simulation Results on Offloading Ratio

The offloading ratio (OR) is defined as the number of subtasks multi-hop offloading to remote edge nodes divided by the total number of subtasks. OR can reflect the behaviour of task offloading and the number of network flows.

For the Alibaba cluster trace dataset, in Fig. 8(a), we observed that the OR increases with the number of tasks, subtasks, and available bandwidth, as well as with the processing speed and number of edge nodes. Similarly, in Fig. 8(b), in the synthetic

(a) Offloading Ratio Results on Alibaba Cluster Trace Dataset

Algorithm	Task Number				Subtask Number				Bandwidth				Computation				Node Number			
	10	20	30	40	25	50	75	100	2	4	6	8	10	20	30	40	25	50	75	100
Random	0.86	0.88	0.89	0.93	0.83	0.84	0.85	0.87	0.81	0.84	0.85	0.87	0.83	0.89	0.9	0.93	0.81	0.84	0.87	0.89
LE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Greedy	0.86	0.88	0.89	0.9	0.9	0.91	0.92	0.96	0.9	0.91	0.93	0.94	0.79	0.80	0.89	0.94	0.91	0.92	0.94	0.95
DQNFCFS	0.86	0.87	0.88	0.89	0.87	0.88	0.97	0.98	0.85	0.87	0.88	0.9	0.81	0.85	0.95	0.98	0.93	0.94	0.96	0.99
APDRL	0.91	0.94	0.96	0.99	0.96	0.97	0.98	0.99	0.94	0.96	0.97	0.99	0.89	0.9	0.95	0.99	0.96	0.97	0.98	0.99

(b) Offloading Ratio Results on Synthetic Dataset

Algorithm	Task Number				Subtask Number				Bandwidth				Computation				Node Number			
	10	20	30	40	25	50	75	100	2	4	6	8	10	20	30	40	25	50	75	100
Random	0.81	0.83	0.83	0.84	0.8	0.86	0.86	0.88	0.79	0.81	0.83	0.84	0.86	0.86	0.86	0.86	0.65	0.71	0.79	0.87
LE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Greedy	0.87	0.88	0.89	0.89	0.89	0.89	0.89	0.89	0.86	0.89	0.93	0.96	0.83	0.86	0.88	0.89	0.72	0.75	0.79	0.93
DQNFCFS	0.88	0.88	0.88	0.87	0.87	0.87	0.87	0.88	0.82	0.85	0.87	0.89	0.81	0.83	0.84	0.87	0.88	0.91	0.97	0.97
APDRL	0.54	0.7	0.77	0.87	0.54	0.57	0.61	0.64	0.75	0.77	0.8	0.95	0.71	0.76	0.77	0.79	0.96	0.97	0.99	0.99

Fig. 8. Simulation reult on offloading ratio.

dataset, the trend remains consistent with the increase in task and subtask numbers, as well as bandwidth and processing speed. However, the OR shows more variability across different algorithms, highlighting the impact of task characteristics on offloading decisions.

E. Influence of Change in the Number of Tasks

1) Simulation Results for Alibaba Cluster Trace Dataset: Fig. 9(a) shows the effect of changing the number of tasks from 10 to 40 on ACT. As shown in Fig. 8(a), the offloading ratio of DQN+FCFS arises with the number of tasks increasing from 10 to 40. Consequently, the increase in ACT for DQN+FCFS is due to both increased network flows and computation loads. There is a significant difference in ACT between ADPRL (LO) and Random. The ACT difference between LE and ADPRL (LO) decreases from 63.3% at 20 tasks to 60.5% at 40 tasks. When the number of tasks increases to 40, ADPRL (LO) achieves an average delay of 40.3% lower than those of DQN + FCFS and Greedy.

Fig. 10(a) illustrates the impact of increasing the number of tasks from 10 to 40 on energy consumption. All algorithms' energy consumption increases with the increasing number of tasks. This increase is primarily attributed to the increasing number of network flows and computation loads as it manages a higher volume of tasks. A significant difference in energy consumption is evident between the ADPRL (EE) and Random strategies. ADPRL (EE) outperforms the Random strategy due to its efficient bandwidth allocation and optimized task scheduling. Specifically, the energy consumption disparity between LE and ADPRL (EE) increases from approximately 10.4% at 20 tasks to around 35.6% at 40 tasks. Notably, when the task count reaches 40, ADPRL (EE) demonstrates a lower energy consumption, averaging 19.6% less than that of the DQN+FCFS and Greedy strategies. This is due to ADPRL (EE)'s advanced learning-based optimization, which becomes increasingly effective as the number of tasks increases.

2) Simulation Results for Synthetic Dataset: As shown in Fig. 9(b), there is an increase in ACT for all algorithms due to both increases in waiting time at the devices to execute the subtasks and the total number of network flows. The increase of network flows can be proved by the rising OR in most

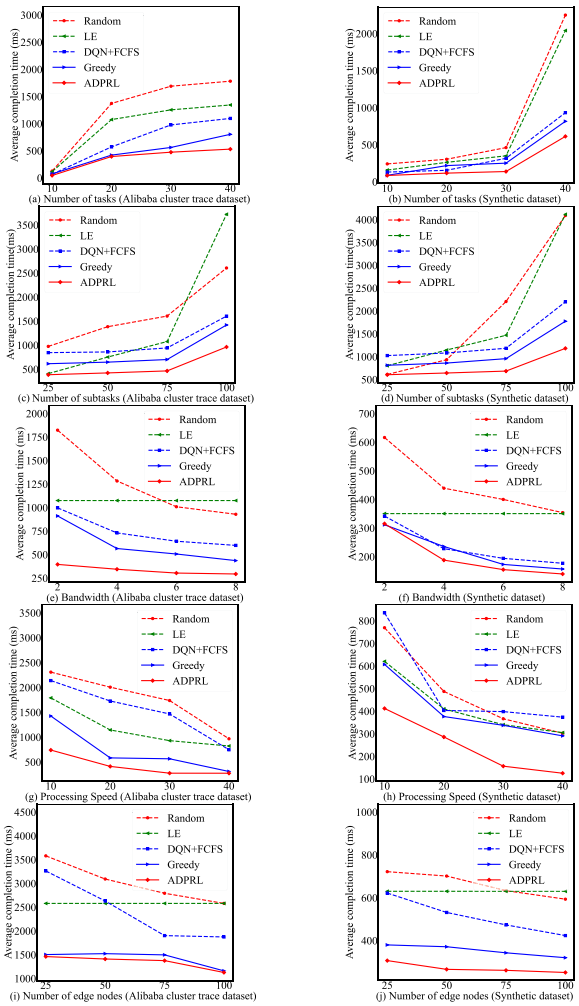


Fig. 9. Simulation results for average completion time.

algorithms from 10 tasks to 40 tasks in Fig. 8(b). ADPRL (LO) is significantly better than Random and LE. Since DQN + FCFS cannot learn an effective task offloading policy, it obtains around 26.4% higher than the ADPRL (LO) at task 40. ADPRL (LO) is around 10% lower than Greedy in terms of ACT.

In Fig. 10(b), ADPRL (EE) demonstrates superior performance compared to the baselines on EC. As the number of tasks increases from 10 to 40, the EC of ADPRL (EE) is consistently lower than that of the other strategies, highlighting its efficiency. Specifically, at 40 tasks, ADPRL (EE)'s EC is approximately 68.4% less than Random, about 63.0% less than LE, around 16.5% less than DQN+FCFS, and about 45.3% lower compared to Greedy.

F. Influence of Change in the Number of Subtasks

1) Simulation Results for Alibaba Cluster Trace Dataset: Fig. 9(c) shows the effect of changing the number of subtasks from 25 to 100 in a task. ADPRL (LO) is significantly better than Random. For LE, tasks are executed at the devices where they are generated, which leads to a longer waiting time as the number of subtasks increases. There is a significant difference in ACT

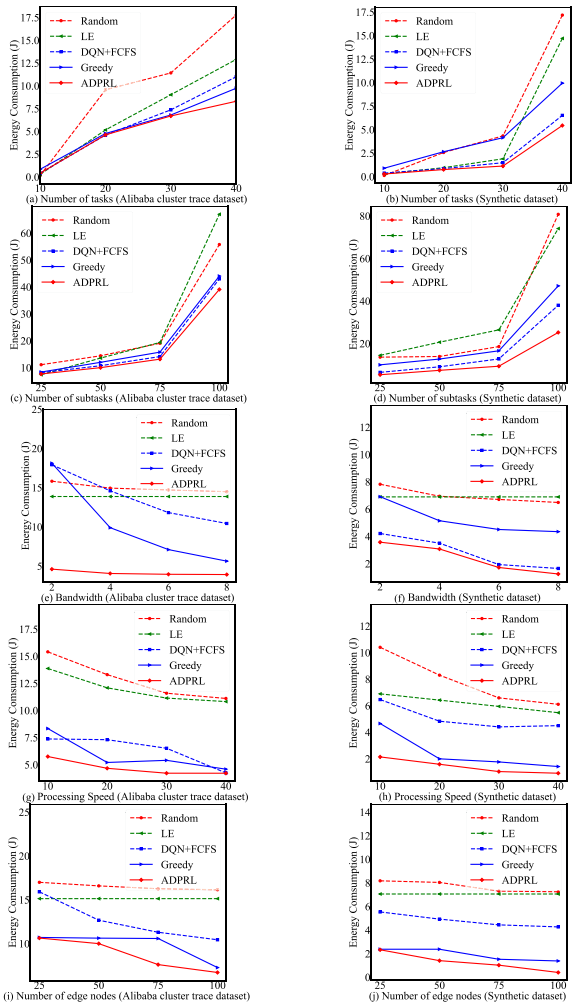


Fig. 10. Simulation results for energy consumption.

between ADPRL (LO) and LE from about 10% at 25 subtasks to 74.1% at 100 subtasks. The difference in ACT between ADPRL (LO) and DQN + FCFS is 40.0% at 100 subtasks because DQN + FCFS does not perform well with a large number of network flows. It can be strengthened by Fig. 8(a), which shows the OR of both ADPRL (LO) and DQN + FCFS is above 90.0% at 100 subtasks. ADPRL (LO) is around 35.9% better than Greedy.

Fig. 10(c) demonstrates the impact of increasing the number of subtasks from 25 to 100 on EC. All algorithms exhibit an increase in EC with the growing number of subtasks. This rise in EC is mainly due to the augmented computational load and network flows required to handle the increasing subtask number. Compared to Random, ADPRL (EE) exhibits up to 29.9% less energy consumption at 100 subtasks. The EC disparity between LE and ADPRL (EE) increases from about 26.2% at 50 tasks to around 41.7% at 100 tasks. ADPRL (EE) demonstrates an approximate average reduction of 9.15% compared to the DQN+FCFS and Greedy strategies. This energy saving is due to ADPRL (EE)'s ability to optimize task execution and bandwidth allocation, reducing unnecessary computational efforts and data transmissions. As a result, ADPRL (EE) consumes less energy while improving task completion time.

2) Simulation Results for Synthetic Dataset: In Fig. 9(d), ADPRL (LO) is significantly better than both Random and LE. Since DQN + FCFS schedules task offloading and network flows separately, it obtains around 11.8% higher than Greedy when the number of subtasks increases to 100. Since ADPRL (LO) can effectively leverage task dependency into a deep deterministic policy gradient, it achieves an average delay of 25.2% lower than those of DQN + FCFS and Greedy at 100 subtasks.

In Fig. 10(d), ADPRL (EE) maintains a consistently lower energy consumption compared to the baselines with the number of subtasks from 25 to 100. At 100 subtasks, ADPRL (EE)'s energy consumption is 68.8% less than Random, 66% less than LE, and 46.5% less than Greedy, while also maintaining a relatively slighter edge of 33.4% less energy consumption compared to DQN+FCFS.

G. Influence of Change in Network Bandwidth

1) Simulation Results for Alibaba Cluster Trace Dataset: Fig. 9(e) shows the effect of changing the average bandwidth value in edge links from 2 Mbps to 8 Mbps on ACT of tasks. The ACT of LE is constant because all tasks are run on edge nodes where they are generated, which can be strengthened by the OR of LE remaining at 0 in Fig. 8(a). It is expected that ADPRL (LO) performs significantly better than baselines in the low bandwidth condition while sharing similar results with baselines in the high bandwidth condition. There is a significant difference in ACT performance between ADPRL (LO) and Random. Greedy is not able to make full use of computation and bandwidth resources when offloading tasks to target nodes. There is an increase in ACT performance difference between ADPRL (LO) and Greedy from about 10% at the average bandwidth 8 Mbps to 56.6% at the average bandwidth 2 Mbps. ADPRL (LO) is around 44.8% less than DQN + FCFS in terms of ACT.

In Fig. 10(e), EC of ADPRL (EE) decreases as bandwidth increases from 2 Mbps to 8 Mbps. This decrease can be attributed to the enhanced efficiency in data transmission at higher bandwidths, reducing the time devices spend in energy-consuming states. The EC of LE is constant due to no network flows. Additionally, when comparing ADPRL (EE) with other algorithms, it is evident that ADPRL (EE) is more energy efficient. Specifically, at the highest bandwidth of 8 Mbps, ADPRL (EE)'s EC is about 73% less than Random, around 62.4% less than DQN+FCFS, and approximately 30.4% less than Greedy. By efficiently managing network resources, ADPRL (EE) minimizes energy waste, thus consuming less energy than the other methods.

2) Simulation Results for Synthetic Dataset: Fig. 9(f) shows the effect of changing the average bandwidth value in edge links from 2 Mbps to 8 Mbps on the ACT of tasks. The ACT of LE is constant because all tasks are run on edge nodes where they are generated. When the bandwidth increases to 8 Mbps, the proposed algorithm ADPRL (LO) achieves an average delay of 43.1% lower than those of Random and LE. Given that the number of network flows remains relatively constant, low bandwidth leads to high latency. Conversely, high bandwidth can substantially decrease latency. Since DQN + FCFS schedules

tasks and network flows separately, which can hardly adapt to dynamic networks, there is an increase in ACT performance difference between ADPRL (LO) and DQN + FCFS from about 10% at the average bandwidth 2 Mbps to 21.13% at the average bandwidth 8 Mbps. ADPRL (LO) is around 10% less than Greedy in terms of ACT.

Fig. 10(f) shows the effect of changing the average value of bandwidth in edge links from 2 Mbps to 8 Mbps on the EC of tasks. The EC of LE remains constant as all tasks are executed on edge nodes where they are generated. When the bandwidth increases to 8 Mbps, ADPRL (EE) achieves an average EC that is 82% lower than both Random and LE. The EC gap between ADPRL (EE) and Greedy increases from approximately 48.3% at an average bandwidth of 2 Mbps to about 71.4% at 8 Mbps. ADPRL (EE) is also around 15.8% less than DQN+FCFS, showcasing the adaptability of ADPRL (EE) to dynamic network conditions.

H. Influence of Change in Processing Speed

1) Simulation Results for Alibaba Cluster Trace Dataset: Fig. 9(g) shows the effect of changing the average value of processing speed in edge links from 10 Mcps to 40 Mcps on ACT of tasks. It is expected that the ACT gap between ADPRL (LO) and baselines will decrease as the processing speed condition transitions from low to high. There is a significant difference in ACT performance between ADPRL (LO) and Random. Greedy cannot make full use of computation and processing speed resources when offloading tasks to target nodes. There is an increase in ACT performance difference between ADPRL (LO) and Greedy from about 11.3% at the average processing speed of 40 Mcps to 48.0% at the average processing speed of 10 Mcps. ADPRL (LO) is around 71.5% less than DQN + FCFS on ACT.

Fig. 10(g) shows a trend of all algorithms decreasing EC as processing speed increases from 10 Mcps to 40 Mcps. All algorithms demonstrate a decrease in EC with faster processing speeds. This decrease is attributed to more efficient task processing, leading to reduced active time and energy usage of devices. Furthermore, ADPRL (EE)'s EC is more pronounced compared to other baselines like Random, LE, Greedy, and DQN+FCFS. Specifically, at 20 Mcps, ADPRL (EE)'s energy consumption is approximately 62.7% less than Random, around 58.5% less than LE, about 22% lower than DQN+FCFS, and nearly 31% less than Greedy. By efficiently managing computational resources through optimized offloading decisions, ADPRL (EE) minimizes energy waste, thus consuming less energy than the other methods.

2) Simulation Results for Synthetic Dataset: Fig. 9(h) shows the effect of changing the average value of processing speed in edge links from 10 Mcps to 40 Mcps on the ACT of tasks. When the processing speed increases to 40 Mcps, the proposed algorithm ADPRL (LO) achieves an average delay of 58.9% lower than those of Random and LE. ADPRL (LO) demonstrates a decrease of around 51.7% in ACT compared to the DQN+FCFS. ADPRL (LO) is around 41.6% less than Greedy in terms of ACT.

Fig. 10(h) shows the energy consumption trend as processing speed increases from 10 Mcps to 40 Mcps. Specifically, at 40 Mcps, ADPRL (EE) shows remarkable energy efficiency compared to the baselines. ADPRL (EE) is approximately 85.1% less than Random, around 83.4% less than LE, about 79.8% less than DQN+FCFS, and nearly 35.7% less than Greedy on energy consumption. ADPRL (EE)'s efficient computational resource management and optimized offloading decisions result in significantly lower energy consumption than other methods.

I. Influence of Change in the Number of Edge Nodes

1) Simulation Results for Alibaba Cluster Trace Dataset: Fig. 9(i) shows the effect of changing the number of edge nodes from 25 to 100 on the average completion time of tasks. It is expected that an increase in the number of edge nodes decreases the average completion time due to the availability of more resources. Given that task and data generation are assumed to be confined to the first 25 nodes, the ACT of LE is constant. It is expected that ADPRL (LO) performs significantly better than baselines. There is a significant difference in ACT performance between ADPRL (LO) and Random. Greedy is not able to make full use of computation and bandwidth resources when offloading tasks to target nodes. There is an average ACT performance difference of 24.8% between ADPRL (LO) and Greedy at the number of edge nodes ranging from 25 to 100. ADPRL (LO) is around 44.8% less than DQN + FCFS in terms of ACT.

Fig. 10(i) reveals a trend of decreasing EC for all algorithms except LE as the number of edge nodes increases from 25 to 100. This is due to a more balanced task distribution across the network, reducing the workload and active time for individual nodes, which consequently lowers energy consumption. However, it's important to consider the tradeoff, as excessively increasing the number of edge nodes may not yield proportional gains in energy efficiency and task completion time. Given that task and data generation are assumed to be confined to the first 25 nodes, the number of edge nodes does not impact the EC of LE. Specifically, at 100 nodes, ADPRL (EE)'s EC is approximately 58.4% less than Random, around 35.8% less than DQN+FCFS, and about 7.7% lower compared to Greedy. These differences illustrate ADPRL (EE)'s effective task allocation and resource management strategies, which optimize task distribution across multiple nodes, minimizing energy waste while efficiently utilizing computational resources.

2) Simulation Results for Synthetic Dataset: As shown in Fig. 9(j), there is a decrease in ACT for all algorithms due to an increase in the number of edge nodes. Given that task and data generation are assumed to be confined to the first 25 nodes, the ACT of LE is constant. ADPRL (LO) is significantly better than Random and LE. The ACT of DQN + FCFS is around 36.4% higher than that of ADPRL (LO). ADPRL (LO) is around 11% lower than Greedy in terms of ACT.

Fig. 10(j) shows the effect of changing the number of edge nodes from 25 to 100 on EC of tasks in the synthetic dataset. Given that task and data generation are assumed to be confined to the first 25 nodes, the EC of LE is constant. Specifically, at 100 nodes, ADPRL (EE) demonstrates remarkable energy efficiency compared to the baselines. It consumes approximately

94.5% less energy than Random, about 94.3% less than LE, about 90.6% less than DQN+FCFS, and nearly 70.8% less than Greedy. ADPRL (EE) effectively utilizes the increased number of edge nodes to distribute tasks more evenly, thus reducing the energy consumption on individual nodes and the overall network.

3) Challenges With Increasing Number of Edge Nodes: There are two main challenges associated with our approach, especially as the number of edge nodes scales from 25 to 100, and even more nodes. One challenge is increased signal transmission delays. As the number of edge nodes increases, the volume of common signals transmitted between the central control and the edge nodes also escalates. This increase in signal traffic results in higher transmission delays, potentially affecting the overall efficiency of the CEC system. Another challenge is the slower convergence of model training. This is due to the more complex and varied data being processed and the need for more extensive coordination and synchronization across nodes.

While these challenges are crucial to the scalability and performance of CEC systems, our current study has certain limitations in directly addressing them. Our work primarily focused on optimizing task offloading, bandwidth allocation, and waiting time decisions under small-scale networks and nodes. In our future studies, we will explore efficient data compression techniques and optimize network protocols to mitigate increased signal transmission delays. For slower model convergence, we will consider advanced parallel training algorithms to accelerate the model training.

J. Performance Analysis Across Different Datasets

The primary reason for employing both the Alibaba cluster trace dataset and the synthetic dataset was to showcase the adaptability and effectiveness of our algorithm across different scenarios. Both the Alibaba cluster trace dataset and the synthetic dataset exhibit a trend where ACT and EC increase as the number of tasks or subtasks rises and decrease as bandwidth, processing speed, and edge node number increase. However, the specific factors contributing to this trend vary between the datasets.

This similarity of performance on the Alibaba cluster trace and synthetic dataset arises primarily due to common distribution properties and modifications to the Alibaba cluster trace dataset. Alibaba cluster trace dataset and synthetic dataset exhibit similar distribution characteristics in some attributes, such as data size and release time. The Alibaba cluster trace dataset, while comprehensive, lacks specific information crucial for our experiments. To address this, we modified the dataset, particularly in attributes like data size and release time, to generate DAGs suitable for our analysis. These modifications, while necessary, led to a certain level of uniformity in the dataset's behavior, aligning it more closely with the Synthetic Data. For instance, the release time of each task of two datasets is randomly selected using a Poisson distribution. The data size of each task is generated randomly from a normal distribution with a mean of 500 Mbit and a variance of 80%.

However, there are performance variations observed between the two datasets. In the Alibaba cluster trace dataset, the rise in ACT is primarily due to increased network flows. For instance, as shown in Fig. 9(c), as the number of subtasks increases from 75 to 100, ACT sharply rises due to a dramatic increase in network flows. Conversely, in the synthetic dataset, ACT is influenced by device waiting time for executing tasks and the total number of network flows. For example, in Fig. 9, with an increasing task number and subtask number, the ACT trend in the synthetic dataset differs from the trend on the Alibaba cluster trace dataset. It initially shows a slow increase followed by a rapid escalation. This reflects the distinct nature of task characteristics in this dataset. The difference can be attributed to the controlled, synthetic nature of the dataset, which allows for a more varied and extensive range of task characteristics.

VII. Conclusion

In this work, we study the ODTO problem and propose ADPRL, an asynchronous DRL-based algorithm. We evaluate ADPRL on simulation experiments with the Alibaba cluster trace dataset and synthetic dataset. The simulation results demonstrate that ADPRL outperforms both heuristic and learning-based baselines. In future work, we will focus on developing strategies to improve the scalability of our approach, particularly by reducing signal transmission delays and accelerating model training.

References

[1] L. U. Khan, I. Yaqoob, N. H. Tran, S. A. Kazmi, T. N. Dang, and C. S. Hong, "Edge-computing-enabled smart cities: A comprehensive survey," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 10200–10232, Oct. 2020.

[2] M. Xu et al., "A full dive into realizing the edge-enabled metaverse: Visions, enabling technologies, and challenges," *IEEE Commun. Surv. Tut.*, vol. 25, no. 1, pp. 656–700, 2022.

[3] 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.

[4] X. Chen, J. Cao, Z. Liang, Y. Sahni, and M. Zhang, "Digital twin-assisted reinforcement learning for resource-aware microservice offloading in edge computing," in *Proc. IEEE 20th Int. Conf. Mobile Ad Hoc Smart Syst.*, 2023, pp. 28–36.

[5] Y. Zhan, S. Guo, P. Li, and J. Zhang, "A deep reinforcement learning based offloading game in edge computing," *IEEE Trans. Comput.*, vol. 69, no. 6, pp. 883–893, Jun. 2020.

[6] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, "Fast adaptive task offloading in edge computing based on meta reinforcement learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 1, pp. 242–253, Jan. 2021.

[7] M. Tang and V. W. 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.

[8] J. Zou, T. Hao, C. Yu, and H. Jin, "A3C-DO: A regional resource scheduling framework based on deep reinforcement learning in edge scenario," *IEEE Trans. Comput.*, vol. 70, no. 2, pp. 228–239, Feb. 2021.

[9] K. Guo, M. Yang, Y. Zhang, and J. Cao, "Joint computation offloading and bandwidth assignment in cloud-assisted edge computing," *IEEE Trans. Cloud Comput.*, vol. 10, no. 1, pp. 451–460, Jan.-Mar. 2022.

[10] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 1, pp. 856–868, Jan. 2019.

[11] A. Gholami and J. S. Baras, "Collaborative cloud-edge-local computation offloading for multi-component applications," in *Proc. IEEE/ACM Symp. Edge Comput.*, 2021, pp. 361–365.

[12] P. Dai, K. Hu, X. Wu, H. Xing, and Z. Yu, "Asynchronous deep reinforcement learning for data-driven task offloading in MEC-empowered vehicular networks," in *Proc. IEEE Conf. Comput. Commun.*, 2021, pp. 1–10.

[13] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.
[14] C.-C. Hung, G. Ananthanarayanan, L. Golubchik, M. Yu, and M. Zhang, "Wide-area analytics with multiple resources," in *Proc. 13th EuroSys Conf.*, 2018, pp. 1–16.
[15] J. Yan, S. Bi, Y. J. Zhang, and M. Tao, "Optimal task offloading and resource allocation in mobile-edge computing with inter-user task dependency," *IEEE Trans. Wireless Commun.*, vol. 19, no. 1, pp. 235–250, Jan. 2020.
[16] S. Sundar and B. Liang, "Offloading dependent tasks with communication delay and deadline constraint," in *Proc. IEEE Conf. Comput. Commun.*, 2018, pp. 37–45.
[17] J. Meng, H. Tan, C. Xu, W. Cao, L. Liu, and B. Li, "Dedas: Online task dispatching and scheduling with bandwidth constraint in edge computing," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 2287–2295.
[18] Z. Han et al., "OnDisc: Online latency-sensitive job dispatching and scheduling in heterogeneous edge-clouds," *IEEE/ACM Trans. Netw.*, vol. 27, no. 6, pp. 2472–2485, Dec. 2019.
[19] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 587–597, Mar. 2018.
[20] J. Wang, J. Hu, G. Min, W. Zhan, A. Zomaya, and N. Georgalas, "Dependent task offloading for edge computing based on deep reinforcement learning," *IEEE Trans. Comput.*, vol. 71, no. 10, pp. 2449–2461, Oct. 2022.
[21] Y. Wang, W. Fang, Y. Ding, and N. Xiong, "Computation offloading optimization for UAV-assisted mobile edge computing: A deep deterministic policy gradient approach," *Wireless Netw.*, vol. 27, no. 4, pp. 2991–3006, 2021.
[22] S. Bi, L. Huang, H. Wang, and Y.-J. A. Zhang, "Lyapunov-guided deep reinforcement learning for stable online computation offloading in mobile-edge computing networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 11, pp. 7519–7537, Nov. 2021.
[23] Y. Sahni, J. Cao, L. Yang, and Y. Ji, "Multihop offloading of multiple DAG tasks in collaborative edge computing," *IEEE Internet Things J.*, vol. 8, no. 6, pp. 4893–4905, Mar. 2021.
[24] M. Zhang, J. Cao, Y. Sahni, Q. Chen, S. Jiang, and L. Yang, "Blockchain-based collaborative edge intelligence for trustworthy and real-time video surveillance," *IEEE Trans. Ind. Informat.*, vol. 19, no. 2, pp. 1623–1633, Feb. 2023.
[25] J. Tang, S. Liu, B. Yu, and W. Shi, "Pi-edge: A low-power edge computing system for real-time autonomous driving services," 2018, *arXiv: 1901.04978*.
[26] S. Ahmad et al.. Aral, "Sustainable environmental monitoring via energy and information efficient multi-node placement," *IEEE Internet Things J.*, vol. 10, no. 24, pp. 22065–22079, Dec. 2023.
[27] E. Angelelli, R. Mansini, and M. G. Speranza, "Kernel search: A general heuristic for the multi-dimensional knapsack problem," *Comput. Operations Res.*, vol. 37, no. 11, pp. 2017–2026, 2010.
[28] B. Jacob et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 2704–2713.
[29] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.
[30] W. Gong, W. Zhang, M. Bilal, Y. Chen, X. Xu, and W. Wang, "Efficient web APIs recommendation with privacy-preservation for mobile app development in industry 4.0," *IEEE Trans. Ind. Informat.*, vol. 18, no. 9, pp. 6379–6387, Sep. 2022.

Xiangchun Chen received the BEng degree in computer science and technology from the Harbin Institute of Technology, China, in 2021. He is currently working toward the PhD degree with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong SAR, China. His research interests lie in digital twin, edge computing, edge AI, and Internet of Things.

Jiannong Cao (Fellow, IEEE) is currently the Otto Poon Charitable Foundation professor in Data Science and the Chair professor of Distributed and Mobile Computing in the Department of Computing with the Hong Kong Polytechnic University (PolyU), Hong Kong. He is also the Dean of Graduate School, the Director of the Research Institute for Artificial Intelligence of Things in PolyU, and the director of the Internet and Mobile Computing Lab. He was the founding director and is now the associate director of PolyU's University Research Facility in Big Data Analytics. He served as the department head from 2011 to 2017. He is a member of Academia Europaea, a fellow of the China Computer Federation (CCF), and an ACM distinguished member. His research interests include distributed systems and blockchain, wireless sensing and networking, Big Data and machine learning, and mobile cloud and edge computing.

Yuvraj Sahni (Member, IEEE) received the BE (Hons) degree in electrical and electronics engineering from the Birla Institute of Technology and Science, Pilani, India, in 2015 and the PhD degree from the Department of Computing, The Hong Kong Polytechnic University, Hong Kong, in 2021. He is currently a research assistant professor with the Department of Building Environment and Energy Engineering, The Hong Kong Polytechnic University, Hong Kong. Before that, he was a postdoctoral fellow with the Department of Computing, The Hong Kong Polytechnic University from 2021 to 2022. His research interests include wireless sensor networks, edge computing, Internet of Things, and smart buildings.

Shan Jiang received the BSc degree in computer science and technology from Sun Yat-Sen University, Guangzhou, China, in 2015, and the PhD degree in computer science from the Hong Kong Polytechnic University, Hong Kong, in 2021. He is currently a research assistant professor with the Department of Computing, The Hong Kong Polytechnic University. Before that, he visited Imperial College London from November 2018 to March 2019. His research interests include distributed systems and blockchain, blockchain-based Big Data sharing, and blockchain as a service. He was the recipient of the best paper award of BlockSys 2021.

Zhixuan Liang received the bachelor's degree in Shenzhen University and master's degree in the Hong Kong Polytechnic University. He is a PhD of the department of computing, the Hong Kong Polytechnic University. His research interests are machine learning, multi-agent reinforcement learning.