TF-DDRL: A Transformer-Enhanced Distributed DRL Technique for Scheduling IoT Applications in Edge and Cloud Computing Environments

Zhiyu Wang, Mohammad Goudarzi, and Rajkumar Buyya, *Fellow, IEEE*

***Abstract*—With the continuous increase of IoT applications, their effective scheduling in edge and cloud computing has become a critical challenge. The inherent dynamism and stochastic characteristics of edge and cloud computing, along with IoT applications, necessitate solutions that are highly adaptive. Currently, several centralized Deep Reinforcement Learning (DRL) techniques are adapted to address the scheduling problem. However, they require a large amount of experience and training time to reach a suitable solution. Moreover, many IoT applications contain multiple interdependent tasks, imposing additional constraints on the scheduling problem. To overcome these challenges, we propose a Transformer-enhanced Distributed DRL scheduling technique, called TF-DDRL, to adaptively schedule heterogeneous IoT applications. This technique follows the Actor-Critic architecture, scales efficiently to multiple distributed servers, and employs an off-policy correction method to stabilize the training process. In addition, Prioritized Experience Replay (PER) and Transformer techniques are introduced to reduce exploration costs and capture long-term dependencies for faster convergence. Extensive results of practical experiments show that TF-DDRL, compared to its counterparts, significantly reduces response time, energy consumption, monetary cost, and weighted cost by up to 60% , 51% , 56% , and 58% , respectively.**

***Index Terms*—Cloud computing, deep reinforcement learning, distributed systems, edge computing, Internet of Things.**

I. Introduction

THE Internet of Things (IoT) paradigm is rapidly emerging as a transformative force and revolutionizing information technology and connectivity. The proliferation of IoT devices and applications has been exponential, reshaping the way humans interact and perceive their surroundings. Cloud computing, as a major driver of the IoT ecosystem, plays a critical role in the storage and process of the large volume of data generated by IoT devices [1]. However, due to the potentially long physical distance between servers in cloud computing and IoT devices, high latency arises, thereby impeding the effective implementation of real-time IoT applications [2]. In response to these challenges, edge computing, as a decentralized computing paradigm, has emerged to provide the ability to process, store, and intelligently control IoT applications [1]. It has quickly become a popular computing paradigm in the IoT environment, offering substantial solutions in various domains. For instance, in the healthcare sector, edge computing can enable real-time monitoring and diagnosis, facilitating faster and more accurate medical decisions [3]; In smart cities, edge computing can be applied to real-time traffic management, improving traffic efficiency, and reducing congestion [4].

However, the considerable increase of IoT applications and servers within edge and cloud computing environments has brought new challenges, necessitating innovative solutions. First, there is an urgent need to minimize the expected response time of IoT applications to ensure efficient and timely performance [5], [6]. Furthermore, in edge and cloud computing environments, the imperative need to minimize server energy consumption and monetary cost is equally crucial for sustainable and cost-effective operations [7]. Thus, scheduling IoT applications on distributed servers to reduce the response time of IoT applications while simultaneously minimizing server energy consumption and monetary cost has become an important and challenging problem.

Given the inherent complexity of this challenge, which can be characterized as an NP-hard problem [8], various solutions have been explored, including heuristic and rule-based approaches [1]. However, these methods face limitations when dealing with the dynamic and unpredictable characteristics of servers in edge and cloud computing. The performance, utilization, and downtime of servers often lack regularity, and the number of IoT applications and their corresponding resource requirements may exhibit randomness. Additionally, IoT applications typically employ Directed Acyclic Graphs (DAGs) for modeling, where nodes represent tasks and edges signify data communication between related tasks [9]. The dependencies between tasks introduce further complexity to the application scheduling process, rendering heuristic and rule-based solutions ineffective in addressing the scheduling challenges presented by IoT computing environments.

Due to the continuous changes in edge and cloud computing environments, decision-making for scheduling IoT applications must be capable of adaptive updates. Deep Reinforcement Learning (DRL), which combines Reinforcement Learning (RL) with Deep Neural Networks (DNN), offers a promising solution. DRL agents can dynamically learn optimal policies and long-term rewards in a stochastic environment without the need for a prior understanding of the system. However, DRL agents must invest substantial time

Received 2 January 2024; revised 22 November 2024; accepted 5 January 2025. Date of publication 10 January 2025; date of current version 10 April 2025. *(Corresponding author: Zhiyu Wang.)*

Zhiyu Wang and Rajkumar Buyya are with the Quantam Cloud Computing and Distributed Systems (QCLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Parkville, VIC 3052, Australia (e-mail: zhiywang1@student.unimelb.edu.au; rbuyya@unimelb.edu.au).

Mohammad Goudarzi is with the Faculty of Information Technology, Monash University, Clayton, VIC 3800, Australia (e-mail: mohammad.goudarzi@monash.edu).

Digital Object Identifier 10.1109/TSC.2025.3528346

during the exploration phase by collecting extensive and diverse experience trajectories, which are later used to learn optimal policies [10]. Hence, the effectiveness of the DRL technique can be prevented by the high exploration costs and slow convergence speeds, negatively impacting the scheduling of IoT applications in highly heterogeneous and stochastic edge and cloud computing environments.

Existing works have increasingly employed DRL techniques for scheduling IoT applications in edge and cloud environments. However, these works face several significant challenges and limitations in practical deployments. First, most existing studies utilize centralized DRL techniques (e.g., Deep Q-Network (DQN), Deep Deterministic Policy Gradient (DDPG)), which require extensive exploration within complex state spaces to identify optimal scheduling strategies [11]. This can lead to high exploration costs and slow convergence rates, particularly in dynamic and heterogeneous edge and cloud environments where exploration costs are further exacerbated. While some works have explored distributed DRL techniques, they predominantly use Asynchronous Advantage Actor Critic (A3C). Although these works leverage multiple parallel agents to collect experience trajectories, the training process primarily relies on asynchronous updates based on the local experiences of individual agents [12]. This limited global knowledge sharing between agents can result in less efficient convergence and suboptimal overall performance [13]. Second, existing works struggle to capture long-term dependencies between tasks, especially in the context of scheduling IoT applications that involve complex task dependency graphs (i.e., DAGs). This limitation often leads to suboptimal resource utilization and increased task completion times in real-world IoT systems. Third, many existing cost models focus on optimizing only one or two objectives, such as minimizing response time or energy consumption. This narrow focus restricts their effectiveness in addressing the multifaceted challenges of dynamic and heterogeneous computing environments. These challenges highlight the need for a comprehensive solution that can effectively address multiple optimization objectives while efficiently handling task dependencies and leveraging distributed learning in heterogeneous edge and cloud environments.

To address these challenges, we propose a distributed DRL technique, named TF-DDRL, based on the Importance Weighted Actor-Learner Architecture (IMPALA) [13], specifically designed for scheduling IoT applications in edge and cloud computing environments. Unlike A3C-based approaches that rely on parameter sharing, IMPALA facilitates the direct sharing of raw experiences between distributed agents, allowing them to collaboratively learn policies more effectively. This enables TF-DDRL to adaptively optimize scheduling decisions across multiple servers, efficiently handling the dynamic and heterogeneous nature of edge and cloud environments. To achieve comprehensive optimization, we propose a weighted cost model that balances multiple objectives, including response time, energy consumption, and monetary costs. This model addresses the limitations of existing models that typically focus on single objectives, enabling more holistic optimization that better reflects real-world requirements. To further tackle the challenge of capturing complex dependencies between IoT tasks and system states in heterogeneous edge and cloud environments, we integrate the Transformer technique [14]. The self-attention mechanism in the Transformer captures both local and global relationships among diverse state components [15], which is essential for making optimal scheduling decisions in highly dynamic and stochastic environments. Additionally, we incorporate Prioritized Experience Replay (PER) [16] to reduce exploration costs by prioritizing more informative experiences, thereby expediting the learning process. The combined use of Transformer and PER not only accelerates the convergence speed of TF-DDRL but also enhances its capacity to derive more effective scheduling strategies, ultimately optimizing response time, energy consumption, and operational costs.

To the best of our knowledge, this is the first work that integrates distributed DRL with Transformer and PER techniques for IoT application scheduling in edge and cloud environments. The main contributions of this paper are as follows.

- We propose a weighted cost model for scheduling DAG-based IoT applications in edge and cloud computing. The objective is to optimize the response time of the application, the energy consumption of the system, and the monetary cost associated with execution. Also, we customize this weighted cost model to comply with DRL algorithms.
- We propose a distributed DRL technique, called TF-DDRL, to solve the weighted cost optimization problem. It can adaptively learn the optimal scheduling policy in response to changes in the computing environment, including diverse IoT application requests and fluctuations of computing resources.
- We design the network structure of TF-DDRL, integrating advanced techniques including PER and the Transformer. This design can significantly improve the convergence speed of the TF-DDRL, ensuring more efficient and effective model performance.
- To evaluate the performance of TF-DDRL, we carry out extensive practical experiments and employ real IoT applications. Through comparisons with distributed DRL techniques including ApeX-Deep Q-Network (ApeX-DQN) [17] and A3C [18], as well as centralized DRL techniques including Dueling Double DQN-RNN (D3QN-RNN) [19], [20] and Soft Actor-Critic (SAC) [21], we highlight the superior performance of TF-DDRL in terms of convergence speed, optimization cost, scalability, and scheduling overhead.

The remainder of the paper is organized as follows. The related literature is provided in Section II, and Section III details the system model and formulate the scheduling problem. The main concepts of the DRL model are presented in Section III-C. The TF-DDRL is discussed in Section IV. Section V evaluates the performance of TF-DDRL and its counterparts. Finally, the concussion and the future work are provided in Section VI.

II. Related Work

The related works that research IoT application scheduling problems in edge and cloud computing environments are studied. Related work is categorized into two groups: centralized reinforcement learning and distributed reinforcement learning.

A. Centralized Reinforcement Learning

In the category of centralized reinforcement learning, Bansal et al. [22] proposed a Dueling-DQN-based technique to place IoT applications in edge and cloud environments. It aims at optimizing the user-side latency and system energy. Hoang et al. [23] proposed an online resource management framework based on the Actor-Critic framework, which considers the long-term constraints of queue stability and computational delay of the queuing system to minimize the average power consumption of the entire system. Huang et al. [24] focused on the resource allocation problem in edge computing environments. They developed a DQN-based approach to minimize a weighted cost, comprising total energy consumption and task completion delay. Zhao et al. [25] proposed a mobile-aware dependent task offloading scheme based on DDPG, with the aim of minimizing the average response time and the average

TABLE I
A COMPARISON OF OUR WORK WITH EXISTING RELATED WORKS

Work	Application Properties		Architectural Properties					Algorithm Properties						Evaluation
	Task Number	Dependency	IoT Device Layer		Edge/Cloud Layer		Multi-Cloud Layer	Main Technique		Optimization Objectives				
			Real Applications	Request Type	Computing Environment	Heterogeneity				Time	Energy	Finance	Multi Objective	
[22]	Multiple	Dependent	◐	Heterogeneous	Edge and Cloud	Heterogeneous	×	Centralized	Dueling-DQN	✓	✓	×	✓	Simulation
[23]	Single	Independent	○	Homogeneous	Edge	Heterogeneous	×		Actor-Critic	×	✓	×	×	Simulation
[24]	Multiple	Independent	◐	Heterogeneous	Edge	Homogeneous	×		DQN	✓	✓	×	✓	Simulation
[25]	Multiple	Dependent	◐	Heterogeneous	Edge	Heterogeneous	×		DDPG	✓	✓	×	✓	Simulation
[26]	Single	Independent	◐	Heterogeneous	Edge and Cloud	Heterogeneous	×		Double-DQN, PG, and Actor-Critic	✓	×	×	×	Simulation
[27]	Multiple	Dependent	○	Homogeneous	Edge	Homogeneous	×		Dueling-DQN and Double-DQN	✓	×	×	✓	Simulation
[28]	Single	Independent	◐	Homogeneous	Edge	Homogeneous	×		SAC	✓	×	×	×	Simulation
[29]	Single	Independent	◐	Homogeneous	Edge	Homogeneous	×		DQN	✓	×	×	✓	Simulation
[30]	Multiple	Independent	◐	Homogeneous	Edge	Homogeneous	×		DQN	✓	×	×	×	Simulation
[31]	Single	Independent	◐	Homogeneous	Edge	Homogeneous	×		DQN	✓	×	×	×	Simulation
[32]	Multiple	Dependent	◐	Heterogeneous	Edge and Cloud	Heterogeneous	×		DQN	✓	×	×	×	Simulation
[33]	Multiple	Dependent	◐	Heterogeneous	Edge and Cloud	Heterogeneous	×	Distributed	Ape-X DQN	✓	×	×	×	Simulation
[34]	Single	Independent	○	Homogeneous	Edge and Cloud	Homogeneous	×		A3C	✓	×	✓	✓	Simulation
[35]	Multiple	Independent	○	Homogeneous	Edge	Homogeneous	×		A3C	✓	×	×	×	Simulation
[36]	Single	Independent	◐	Heterogeneous	Edge	Heterogeneous	×		A3C	×	✓	×	×	Simulation
[37]	Single	Independent	◐	Heterogeneous	Edge	Heterogeneous	×		A3C	×	✓	×	×	Simulation
[38]	Multiple	Dependent	○	Homogeneous	Edge and Cloud	Heterogeneous	×		A3C	✓	✓	×	✓	Simulation
Our work	Multiple	Dependent	●	Heterogeneous	Edge and Cloud	Heterogeneous	✓		IMPALA	✓	✓	✓	✓	Practical

●: Real IoT Application and Deployment, ◐: Simulated IoT Application, ○: Random

energy consumption of the system. Hsieh et al. [26] investigated the task allocation problem in collaborative Mobile Edge Computing (MEC) networks, developing and comparing the performance of Double-DQN, PG, and Actor-Critic in optimizing delay and task overflow rate. The results demonstrated that the Actor-Critic approach performed the best in dynamic MEC network environments. Fan et al. [27] studied the problem of user task offloading in MEC network environments and proposed a technique based on Dueling-DQN and Double-DQN to optimize response time and dropped task ratio. Zheng et al. [28] defined an optimization problem involving computational offloading and resource allocation in collaborative vehicle networks. A Soft Actor-Critic (SAC)-based technique is proposed to reduce the overall delay of the system. Wang et al. [29] proposed a computing resource allocation solution based on DQN specifically for edge computing environments. The objective of their research is to optimize the average time overhead and achieve a more balanced utilization of resources within edge environments. Xiong et al. [30] aimed at reducing the average job completion time within edge computing environments by employing a DQN-based resource allocation policy. Jie et al. [31] focused on the time overhead optimization problem in edge computing environments and employed a DQN-based method to reduce the task execution time. They formulate the optimization problem as a Markov Decision Process (MDP). [32] proposed a dependency-aware task offloading method with DQN to optimize task offloading in cloud-edge environments. Their approach models mobile applications as DAGs and leverages DQN to adaptively handle dynamic resource changes and parallel task scheduling without presetting task priorities.

B. Distributed Reinforcement Learning

Wen et al. [33] introduced an adaptive scheduler based on environmental changes, aiming to reduce the tail latency of edge-cloud jobs. The work employs Ape-X DQN to expedite the training process. Wang et al. [34] proposed an Asynchronous Advantage Actor-Critic (A3C)-based approach to address the cloud-edge computing network optimization problem, aiming at satisfying the latency requirements of applications while reducing the cost of cloud servers. Garaali et al. [35] investigated the optimization problem of computational offloading and resource allocation in an MEC environment and proposed a solution based on the A3C method. In order to reduce the system latency, each agent aims to learn the optimal offloading policy independently of the other agents in an interactive manner. Ju et al. [36] considered the task offloading problem in vehicular edge computing networks, where the joint optimization is formulated as MDP. This work proposes an A3C-based approach to solve the MDP problem, with the goal of minimizing the system energy consumption while satisfying computational delay constraints. Sellami et al. [37] investigated IoT application scheduling and offloading problems in edge computing environments. This work introduces a scheduling policy based on A3C to enhance energy efficiency. Chen et al. [38] studied the task offloading problem in cloud-edge collaborative mobile computing environments, proposing an A3C-based algorithm to address the joint optimization problem involving task execution delay and energy consumption. Utilizing a distributed learning approach, the algorithm acquires knowledge about the probability distribution of an approximate reward and optimizes network parameters using the computing resource in the cloud, with the objective of achieving faster and more efficient decision-making.

C. A Qualitative Comparison

Table I provides a qualitative analysis of current research work and ours in various dimensions, including application properties, architectural properties, algorithm properties, and evaluation. Application properties explore whether the IoT application has multiple tasks or not and their interdependencies. Architectural properties are divided into three layers. The IoT device layer identifies the practical characteristics of the application and the request type of the IoT devices. The section named real applications provides information on whether the work utilizes real-world IoT applications, simulations, or randomly generated data, and distinct IoT devices with varying quantities of requests and diverse requirements are classified as heterogeneous request types. The edge/cloud layer delves into the computing environment considered by the work and the heterogeneity of deployment servers. Moreover, the multi-cloud layer assesses whether the work takes into account cloud computing resources from different providers. In the algorithm properties section, the focus is on the primary techniques employed by the work and the optimization objectives. Finally, the evaluation section

TABLE II
LIST OF KEY NOTATIONS

Variable	Description	Variable	Description
$\mathcal{A}$	One application set	$PR(\mathcal{A}_i^j)$	The set of predecessor tasks of task $\mathcal{A}_i^j$
$\mathcal{A}_i$	One application	$SU(\mathcal{A}_i^j)$	The set of successor tasks of task $\mathcal{A}_i^j$
$\mathcal{A}_i^j$	One single task	T	The response time model
N	The server set	T^{dat}	The Data Arrival Time (DAT) model
$\mathcal{CS}$	The cloud server set	T^{ex}	The execution time model
$\mathcal{ES}$	The edge server set	T^{tr}	The transmission time model
N_k	One single server	E	The energy consumption model
$Freq(\mathcal{N}_k)$	The available CPU frequency (MHz) of server $\mathcal{N}_k$	E^{ex}	The execution energy model
$Ram(\mathcal{N}_k)$	The available RAM size (GB) of server $\mathcal{N}_k$	E^{tr}	The transmission energy model
$\mathcal{C}_i$	The scheduling configuration for application $\mathcal{A}_i$	$W^{ex}(\mathcal{N}_k)$	The power of the server $\mathcal{N}_k$ when executing task
$\mathcal{C}_i^j$	The scheduling configuration for task $\mathcal{A}_i^j$	$W^{tr}(\mathcal{N}_k)$	The power of the server $\mathcal{N}_k$ when transmitting data
$\mathcal{P}_{\mathcal{N}_l,\mathcal{N}_k}$	The propagation time (ms) between server $\mathcal{N}_l$ and server $\mathcal{N}_k$	F	The monetary cost model
$DS_{\mathcal{N}_l,\mathcal{N}_k}(\mathcal{A}_i^j)$	The data size for task $\mathcal{A}_i^j$ sent from server $\mathcal{N}_l$ to server $\mathcal{N}_k$	$CLP(\mathcal{N}_k)$	The pricing of the cloud server $\mathcal{N}_k$
$\mathcal{B}_{\mathcal{N}_l,\mathcal{N}_k}$	The data rate (bits/second) between server $\mathcal{N}_l$ and server $\mathcal{N}_k$	$EP(\mathcal{N}_k)$	The electricity price for running edge server $\mathcal{N}_k$
$L(\mathcal{A}_i^j)$	The required CPU cycles for task $\mathcal{A}_i^j$	J	The weighted cost model

determines whether the work is evaluated through simulation or practical applications.

In the literature, many works (e.g., [23], [24], [26], [28], [29], [30], [31], [34], [35], [36], and [37]) assume that tasks are mutually independent and do not consider the common occurrence of task dependencies in the real world. However, in practical scenarios, such dependencies significantly impact scheduling performance. For example, in smart city traffic management, vehicle detection relies on real-time video data collection, while traffic flow prediction depends on the results of vehicle detection. Neglecting these dependencies can lead to processing delays and reduced system efficiency. Similarly, in healthcare, initial patient data processing must be completed before cloud-based analysis, or else diagnoses could be delayed. In industrial IoT, equipment fault detection is closely linked to subsequent maintenance tasks, where improper scheduling could result in production disruptions. Also, works including [22], [23], [24], [25], [26], [27], [28], [29], [30], [31], and [32], adopt centralized DRL techniques, which may incur high exploration costs and exhibit low convergence speed [39]. This poses challenges when deployed in highly distributed computing environments, especially as the number of features, environmental complexity, and application constraints increase. Furthermore, most of the work employing distributed DRL techniques is based on A3C, including [34], [35], [36], [37], and [38]. Despite deploying distributed agents to collect experience trajectories, distributed agents in A3C train their local policies based on their limited experiences, subsequently forwarding these parameters to learners for aggregation and training, diminishing the usage efficiency of experience trajectories. To address these issues, we propose a distributed DRL technique, called TF-DDRL, which learns policies based on direct sharing of original experience, rather than parameters. Besides, TF-DDRL employs PER to enhance sampling efficiency and incorporates the Transformer to capture long-term dependencies between features, further improving convergence speed and optimizing performance. Moreover, our work considers inter-task dependencies when addressing IoT application scheduling problems in edge and multi-cloud heterogeneous environments. Also, we establish a practical experimental environment employing both real-time and non-real-time IoT applications to evaluate the performance of TF-DDRL.

III. System Model and Problem Formulation

This section first describes the topology of IoT systems in this work. Next, we tackle the scheduling of IoT applications by formulating it as an optimization problem, aiming at reducing application response time, system energy consumption, and monetary cost of running applications. Table II depicts the key notations used in this paper.

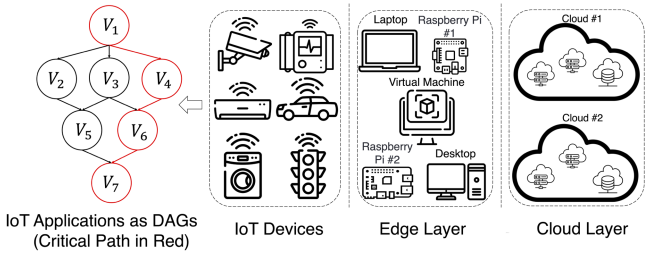


Fig. 1. An overview of Edge and Cloud computing.

A. System Model

Fig. 1 provides a layered perspective of the IoT system in an edge and cloud environment. Consider $\mathcal{A} = \{\mathcal{A}_i | 1 \leq i \leq |\mathcal{A}|\}$ as a collection of $|\mathcal{A}|$ applications, with each application comprising one or more tasks denoted as $\mathcal{A}_i = \{\mathcal{A}_i^j | 1 \leq j \leq |\mathcal{A}_i|\}$. To model an IoT application, we use a DAG, as illustrated in Fig. 1, where each vertex $\mathcal{V}_j = \mathcal{A}_i^j$ corresponds to a specific task within the application $\mathcal{A}_i$. The edges, represented as $\mathcal{E}_{j,k}$, signify the data flow between tasks $\mathcal{V}_j$ and $\mathcal{V}_k$, indicating that successor tasks must follow the completion of their predecessors. Also, the critical path of the DAG, denoted as $CP(\mathcal{A}_i)$ and marked in red in the figure, shows the path with the highest cost.

We consider a server set comprising $|\mathcal{N}|$ servers to handle the application set $\mathcal{A}$, denoted as $\mathcal{N} = \mathcal{CS} \cup \mathcal{ES} = \{\mathcal{N}_k | 1 \leq k \leq |\mathcal{N}|\}$. $\mathcal{CS}$ denotes the cloud server set and $\mathcal{ES}$ denotes the edge server set. To consider server heterogeneity, each server $\mathcal{N}_k$ is characterized by different available CPU frequency (MHz) $Freq(\mathcal{N}_k)$ and available RAM size (GB) $Ram(\mathcal{N}_k)$. Also, we consider $\mathcal{P}_{\mathcal{N}_l,\mathcal{N}_k}$ as the propagation time (ms) and $\mathcal{B}_{\mathcal{N}_l,\mathcal{N}_k}$ as the data rate (b/s) between server $\mathcal{N}_l$ and $\mathcal{N}_k$.

B. Problem Formulation

Since an application consists of one or more tasks, it can run on various servers. Considering server set as $\mathcal{N}$, the scheduling configuration $\mathcal{C}_i^j$ for task $\mathcal{A}_i^j$ is defined as:

$$\mathcal{C}_i^j = \mathcal{N}_k, \quad k \in \{1, \ldots, |\mathcal{N}|\}, \tag{1}$$

where N_k denotes a particular server, and $|N|$ is the total number of servers. This variable serves as the atomic decision unit in the scheduling process, determining the computational resource allocated for each task.

The scheduling configuration $\mathcal{C}_i$ for the application $\mathcal{A}_i$ is a collection of the scheduling configurations for the tasks within $\mathcal{A}_i$,

and is defined as:

$$\mathcal{C}_i = \{\mathcal{C}_i^j | 1 \leq j \leq |\mathcal{A}_i|\}, \tag{2}$$

where $|A_i|$ represents the total number of tasks in application A_i. By grouping task-level scheduling configurations, C_i provides a comprehensive view of how the entire application A_i is distributed across available servers.

Also, the task execution model of one application can exhibit hybrid characteristics, incorporating both sequential and/or parallel processes. Accordingly, each task cannot be executed unless all predecessor tasks complete their execution, while tasks that are not dependent on each other can be executed in parallel. We use $PR(\mathcal{A}_i^j)$ to denote the set of predecessor tasks of task $\mathcal{A}_i^j$ and use $CP(\mathcal{A}_i^j)$ to indicate whether task $\mathcal{A}_i^j$ is located on the critical path of the application $\mathcal{A}_i$.

1) Response Time Model: Assuming that the scheduling configuration for task $\mathcal{A}_i^j$ is $\mathcal{C}_i^j$, the response time model $T(\mathcal{C}_i^j)$ consists of two parts, the Data Arrival Time (DAT) model $T^{dat}(\mathcal{C}_i^j)$ and the execution time model $T^{ex}(\mathcal{C}_i^j)$:

$$T(\mathcal{C}_i^j) = T^{dat}(\mathcal{C}_i^j) + T^{ex}(\mathcal{C}_i^j). \tag{3}$$

The DAT model $T^{dat}(\mathcal{C}_i^j)$ signifies the maximum time for the data, required by task $\mathcal{A}_i^j$, to reach the designated server:

$$T^{dat}(\mathcal{C}_i^j) = \max \ \ T^{dat}_{\mathcal{C}_i^k,\mathcal{C}_i^j}, \quad \forall \mathcal{A}_i^k \in PR(\mathcal{A}_i^j), \tag{4}$$

where $T^{dat}_{\mathcal{C}_i^k,\mathcal{C}_i^j}$ shows the time consumed for the required data to be transmitted from scheduled server $\mathcal{C}_i^k$ to server $\mathcal{C}_i^j$. Here, $\mathcal{C}_i^j$ signifies the server scheduled for the execution of task $\mathcal{A}_i^j$, while $\mathcal{C}_i^k$ corresponds to the server where the predecessor task of task $\mathcal{A}_i^j$ is executed. Thus, $T^{dat}_{\mathcal{C}_i^k,\mathcal{C}_i^j}$ depends on both the transmission time $T^{tr}_{\mathcal{C}_i^k,\mathcal{C}_i^j}$ and the propagation time $\mathcal{P}_{\mathcal{C}_i^k,\mathcal{C}_i^j}$ for task $\mathcal{A}_i^j$ between server $\mathcal{C}_i^k$ and server $\mathcal{C}_i^j$:

$$T^{dat}_{\mathcal{C}_i^k,\mathcal{C}_i^j} = \begin{cases} T^{tr}_{\mathcal{C}_i^k,\mathcal{C}_i^j} + \mathcal{P}_{\mathcal{C}_i^k,\mathcal{C}_i^j} & \mathcal{C}_i^k \neq \mathcal{C}_i^j, \\ 0 & \mathcal{C}_i^k = \mathcal{C}_i^j. \end{cases} \tag{5}$$

where the transmission time $T^{tr}_{\mathcal{C}_i^k,\mathcal{C}_i^j}$ is calculated as follows:

$$T^{tr}_{\mathcal{C}_i^k,\mathcal{C}_i^j} = \frac{DS_{\mathcal{C}_i^k,\mathcal{C}_i^j}(\mathcal{A}_i^j)}{\mathcal{B}_{\mathcal{C}_i^k,\mathcal{C}_i^j}}, \tag{6}$$

$DS_{\mathcal{C}_i^k,\mathcal{C}_i^j}(\mathcal{A}_i^j)$ denotes the data size for task $\mathcal{A}_i^j$ sent from server $\mathcal{C}_i^k$ to server $\mathcal{C}_i^j$, and $\mathcal{B}_{\mathcal{C}_i^k,\mathcal{C}_i^j}$ denotes the current bandwidth between server $\mathcal{C}_i^k$ and server $\mathcal{C}_i^j$.

The execution time model $T^{ex}(\mathcal{C}_i^j)$ is defined as the time required to execute task $\mathcal{A}_i^j$ based on scheduling configuration $\mathcal{C}_i^j$. It can be calculated as follows:

$$T^{ex}(\mathcal{C}_i^j) = \frac{L(\mathcal{A}_i^j)}{Freq(\mathcal{C}_i^j)}, \tag{7}$$

where $L(\mathcal{A}_i^j)$ denotes the necessary CPU cycles for task $\mathcal{A}_i^j$ to be executed and $Freq(\mathcal{C}_i^j)$ shows the CPU frequency of scheduled server $\mathcal{C}_i^j$ (if the CPU has multiple cores, the model considers the average frequency). Accordingly, the formulation of the response time model $T(\mathcal{C}_i)$ for application $\mathcal{A}_i$ is expressed as follows:

$$T(\mathcal{C}_i) = \sum_{j=1}^{|\mathcal{A}_i|} (T(\mathcal{C}_i^j) \times CP(\mathcal{A}_i^j)), \tag{8}$$

where $CP(\mathcal{A}_i^j)$ is the critical path indicator. If task $\mathcal{A}_i^j$ belongs to the critical path of application $\mathcal{A}_i$, $CP(\mathcal{A}_i^j)$ is set to 1; otherwise, it assumes a value of 0.

2) Energy Consumption Model: In this work, we consider the energy consumption of the edge layer and the cloud layer. Given that the scheduling configuration for task $\mathcal{A}_i^j$ is $\mathcal{C}_i^j$, the energy consumption $E(\mathcal{C}_i^j)$ is determined by the energy consumed during the actual task processing (i.e., the execution energy model) $E^{ex}(\mathcal{C}_i^j)$, plus the energy consumed by the servers when transmitting the required data to other servers (i.e., transmission energy model) $E^{tr}(\mathcal{C}_i^j)$:

$$E(\mathcal{C}_i^j) = E^{ex}(\mathcal{C}_i^j) + (E^{tr}(\mathcal{C}_i^j) \times ED(\mathcal{A}_i^j)), \tag{9}$$

where $ED(\mathcal{A}_i^j)$ is 0 if $\mathcal{A}_i^j$ is the ending task (i.e., has no successor task) in application $\mathcal{A}_i$ and 1 otherwise.

The execution energy model $E^{ex}(\mathcal{C}_i^j)$ is the energy consumed by the server to execute the task, defined as:

$$E^{ex}(\mathcal{C}_i^j) = T^{ex}(\mathcal{C}_i^j) \times W^{ex}(\mathcal{C}_i^j), \tag{10}$$

where $T^{ex}(\mathcal{C}_i^j)$ is obtained from (7) and $W^{ex}(\mathcal{C}_i^j)$ represents the power of the server when executing the task.

Considering the dependency between tasks, one task $\mathcal{A}_i^j$ can have one or more predecessor tasks. The transmission energy model $E^{tr}(\mathcal{C}_i^j)$ is defined as the sum of the energy consumed to transmit the data to the servers where the successor tasks are assigned, as follows:

$$E^{tr}(\mathcal{C}_i^j) = \sum_{\mathcal{A}_i^l \in SU(\mathcal{A}_i^j)} \frac{DS_{\mathcal{C}_i^j,\mathcal{C}_i^l}(\mathcal{A}_i^j)}{\mathcal{B}_{\mathcal{C}_i^j,\mathcal{C}_i^l}} \times W^{tr}(\mathcal{C}_i^j) \times OS(\mathcal{C}_i^j, \mathcal{C}_i^l), \tag{11}$$

where $SU(\mathcal{A}_i^j)$ denotes the set of successor tasks of task $\mathcal{A}_i^j$, $\mathcal{C}_i^l$ is the scheduling configuration of task $\mathcal{A}_i^l$, $W^{tr}(\mathcal{C}_i^j)$ show the transmission power of the server when transmitting data, and $OS(\mathcal{C}_i^j, \mathcal{C}_i^l)$ is 0 if $\mathcal{C}_i^j$ and $\mathcal{C}_i^j$ are the same server and 1 otherwise. Similar to [40], [41], $W^{tr}(\mathcal{C}_i^j)$ is set as a constant value, but this parameter can also be dynamically adjusted.

Accordingly, the energy consumption model $E(\mathcal{C}_i)$ for application $\mathcal{C}_i$ is formulated as follows:

$$E(\mathcal{C}_i) = \sum_{j=1}^{|\mathcal{A}_i|} E(\mathcal{C}_i^j), \tag{12}$$

3) Monetary Cost Model: The server set $\mathcal{N}$ comprises both the cloud server set $\mathcal{CS}$ and the edge server set $\mathcal{ES}$. Without loss of generality, we assume that the edge servers are on-premises servers and are owned by users, so their execution cost only depends on their electricity usage. Otherwise, the cloud-like pricing model can be used for edge servers. Given that the scheduling configuration for the task $\mathcal{A}_i^j$ is $\mathcal{C}_i^j$, the monetary cost $F(\mathcal{C}_i^j)$ depends both on the cloud server and the edge server price models. Formally, the

monetary cost model $F(\mathcal{C}_i^j)$ is:

$$F(\mathcal{C}_i^j) = \begin{cases} \sum_{\mathcal{C}_i^j \in \mathcal{CS}} T(\mathcal{C}_i^j) \times CLP(\mathcal{C}_i^j) & \mathcal{C}_i^j \in \mathcal{CS}, \\ \sum_{\mathcal{C}_i^j \in \mathcal{ES}} E(\mathcal{C}_i^j) \times EP(\mathcal{C}_i^j) & \mathcal{C}_i^j \in \mathcal{ES}, \end{cases} \tag{13}$$

where $CLP(\mathcal{C}_i^j)$ shows the cloud server pricing, and $EP(\mathcal{C}_i^j)$ denotes the electricity price for running edge server $\mathcal{C}_i^j$. Consequently, the monetary cost model $F(\mathcal{C}_i)$ for the application $\mathcal{C}_i$ is formulated as follows:

$$F(\mathcal{C}_i) = \sum_{j=1}^{|\mathcal{A}_i|} F(\mathcal{C}_i^j). \tag{14}$$

4) Weighted Cost Model: The weighted cost model $J(\mathcal{C}_i^j)$ is defined as the weighted sum of the normalized response time models, the energy consumption model, and the monetary cost model. Given the scheduling configuration for task $\mathcal{A}_i^j$ is $\mathcal{C}_i^j$:

$$J(\mathcal{C}_i^j) = w_1 \frac{T(\mathcal{C}_i^j) - T^{\min}}{T^{\max} - T^{\min}} + w_2 \frac{E(\mathcal{C}_i^j) - E^{\min}}{E^{\max} - E^{\min}} + w_3 \frac{F(\mathcal{C}_i^j) - F^{\min}}{F^{\max} - F^{\min}}, \tag{15}$$

where $T^{\min}$, $T^{\max}$, $E^{\min}$, $E^{\max}$, $F^{\min}$, and $F^{\max}$ represent the minimum and the maximum value that can be achieved by the response time model, the energy consumption model, and the monetary cost model, respectively. Also, w_1, w_2, and w_3 are the control parameters used to fine-tune the weighted cost model. The reason for employing normalized models, rather than the original models, is that the values of the models may fall within different ranges.

Accordingly, the weighted cost model for application $\mathcal{A}_i$ is defined as:

$$J(\mathcal{C}_i) = w_1 \times Norm(T(\mathcal{C}_i)) + w_2 \times Norm(E(\mathcal{C}_i)) + w_3 \times Norm(F(\mathcal{C}_i)), \tag{16}$$

where $T(\mathcal{C}_i)$, $E(\mathcal{C}_i)$, and $F(\mathcal{C}_i)$ are obtained from (8), (12) and (14), and $Norm$ represents the normalization.

Therefore, the optimization problem of scheduling IoT applications can be formulated as:

$$min \quad J(\mathcal{C}_i) \tag{17}$$

$$\text{s.t.} \quad C1: \ Size(\mathcal{C}_i^j) = 1, \ \forall \mathcal{C}_i^j \in \mathcal{C}_i \tag{18}$$

$$C2: \ DS_{\mathcal{C}_i^k, \mathcal{C}_i^j}(\mathcal{A}_i^j), \mathcal{B}_{\mathcal{C}_i^k, \mathcal{C}_i^j} > 0, \ \forall \mathcal{C}_i^k, \mathcal{C}_i^j \in \mathcal{N}, \ \forall \mathcal{A}_i^j \in \mathcal{A}_i \tag{19}$$

$$C3: \ Freq(\mathcal{N}_k), Ram(\mathcal{N}_k) > 0, \ \forall \mathcal{N}_k \in \mathcal{N} \tag{20}$$

$$C4: \sum_{\mathcal{A}_i \in \mathcal{A}} \sum_{\mathcal{A}_i^j \in \mathcal{A}_i} Ram(\mathcal{A}_i^j) \times SO(\mathcal{A}_i^j, \mathcal{N}_k) < Ram(\mathcal{N}_k), \ \forall \mathcal{N}_k \in \mathcal{N} \tag{21}$$

$$C5: \ T(\mathcal{A}_i^j) \leq T(\mathcal{A}_i^j + \mathcal{A}_i^k), \forall \mathcal{A}_i^j \in PR(\mathcal{A}_i^k) \tag{22}$$

$$C6: \ w_1 + w_2 + w_3 = 1, \ 0 \leq w_1, w_2, w_3 \leq 1 \tag{23}$$

where $C1$ enforces the rule that each task can be assigned to only one server. $C2$ specifies the transmission constraints for data size and bandwidth. Additionally, $C3$ defines constraints related to the available CPU frequency and available RAM size of the server by setting a lower bound. Furthermore, $C4$ ensures that every server has adequate RAM resources to process all tasks scheduled on it, preventing resource overutilization. $SO(\mathcal{A}_i^j, \mathcal{N}_k)$ equals 1 if task $\mathcal{A}_i^j$ is scheduled on server $\mathcal{N}_k$, otherwise 0. $C5$ specifies that each task is eligible for processing only after the completion of its predecessor tasks, ensuring that the accumulative cost is no less than that of the predecessor task. Lastly, $C6$ places restrictions on the control parameters within (16), confining them to values between 0 and 1.

The problem under consideration is characterized as a non-convex optimization problem, primarily due to the potential existence of an infinite number of local optima within the feasible domain. Typically, algorithms aimed at finding the global optimum in such problems exhibit exponential complexity and are classified as NP-hard [42]. To solve such a non-convex optimization problem, most approaches decompose these problems into several convex sub-problems [43], subsequently solving these sub-problems iteratively until convergence is achieved [44]. However, this strategy often sacrifices accuracy for reduced complexity [45]. Also, these approaches are heavily dependent on the current environment and are not suitable for dynamic environments with highly heterogeneous computational resources [46]. To tackle this issue, we propose TF-DDRL to adaptively manage uncertainties in dynamic and stochastic environments. It can dynamically learn scheduling policies through continuous interaction with the environment.

C. Deep Reinforcement Learning Model

To apply the DRL approach, the optimization problem should be formulated as a MDP. More specifically, the problem can be defined by the tuple $< \mathbb{S}, \mathbb{A}, \mathbb{P}, \mathbb{R}, \gamma >$, where $\mathbb{S}$ signifies a finite set of states, $\mathbb{A}$ represents a finite set of actions, $\mathbb{P}$ represents the state transition probability, $\mathbb{R}$ stands for the reward function, and $\gamma \in [0, 1]$ serves as the discount factor employed in calculating cumulative rewards.

We consider the learning process to be divided into multiple time steps t within a total time span $\mathbb{T}$. At each time step, the agent interacts with the environment, resulting in multiple states S_t. At time step t, the agent observes the environment state $S_t = s$, where $s \in \mathbb{S}$. Guided by the policy $\pi(a|s)$, where $a \in \mathbb{A}$, the agent chooses an action $A_t = a$. The policy function $\pi(a|s) = Pr[A_t = a|S_t = s]$ explicitly defines the probability of selecting action a given state s. Following the execution of action a, the agent receives a reward $r = \mathbb{R}[S_t = s, A_t = a]$ from the environment, determined by the reward function $\mathbb{R}$. The agent then undergoes a state transition to $S_{t+1} = s'$ based on the state transition function $P_{ss'}^a = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a]$. The ultimate objective of the agent is to acquire a policy π maximizing the expected cumulative discounted reward, denoted as $\mathbb{E}\pi[\sum t \in T \gamma_t r_t]$.

Considering the scheduling problem of IoT applications in edge and cloud computing environments, the MDP's state space $\mathbb{S}$, action space $\mathbb{A}$, and reward function $\mathbb{R}$ are defined as follows:

- *State space* $\mathbb{S}$: In this work, the formulated problem pertains to tasks and servers, with the state $\mathbb{S}$ containing $\mathbb{F}$ for the task feature and $\mathbb{G}$ for the server set state. At time step t, the feature space $\mathbb{F}$ of task $\mathcal{A}_i^j$ captures essential details related to the task, defined as:

$$\mathbb{F}_t(\mathcal{A}_i^j) = \{f_t^y(\mathcal{A}_i^j) | \mathcal{A}_i^j \in \mathcal{A}_i, 0 \leq y \leq |\mathbb{F}|\}, \tag{24}$$

where y denotes the feature index and $|\mathbb{F}|$ represents the total number of features. Specifically, the feature space $\mathbb{F}$ includes task ID, application ID, required CPU cycles $L(\mathcal{A}_i^j)$, required RAM size $Ram(\mathcal{A}_i^j)$, task dependencies (predecessors $PR(\mathcal{A}_i^j)$ and successors $SU(\mathcal{A}_i^j)$), previously configured tasks and their scheduled servers, execution status of dependent tasks, etc. This comprehensive feature space enables the DRL agent to make informed scheduling decisions.

Also, in time step t, the state space $\mathbb{G}$ of the server set $\mathcal{N}$ contains the number of servers, the CPU frequency, RAM size,

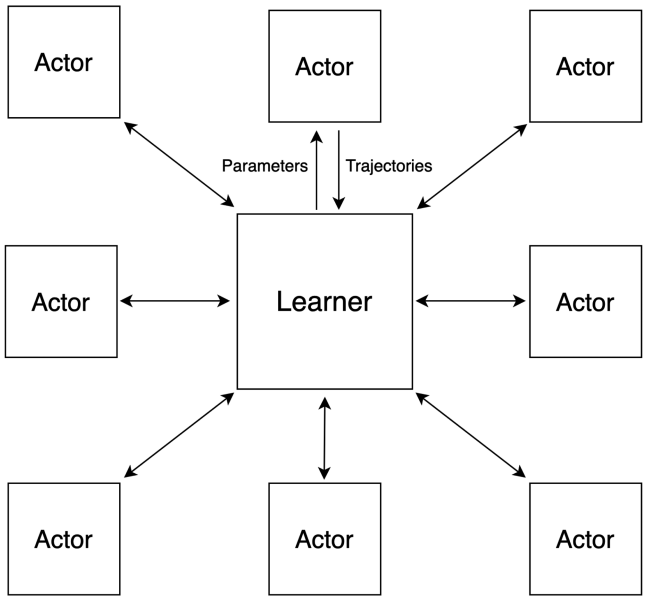


Fig. 2. High-level architecture of Actors and Learner.

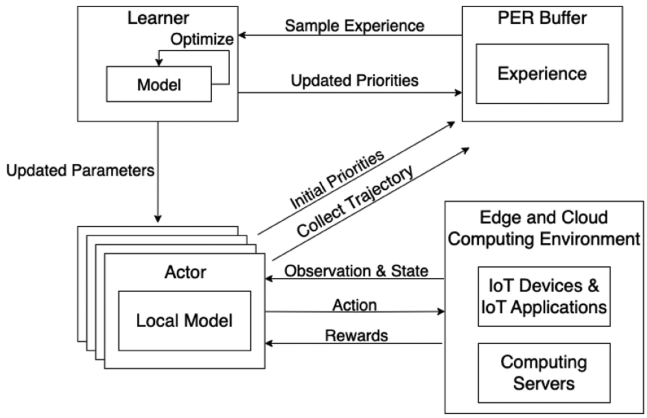


Fig. 3. An overview of TF-DDRL framework.

label (e.g., cloud or edge), expense per time unit (for cloud servers), electricity price (for edge servers), propagation time, bandwidth between different servers, etc, which is formally defined as:

$$\begin{aligned}&\mathbb{G}_t(\mathcal{N}) = \{|\mathcal{N}|, g_t^z(\mathcal{N}_k), h_t^q(\mathcal{N}_j, \mathcal{N}_k)| \\ &\mathcal{N}_j, \mathcal{N}_k \in \mathcal{N}, 0 \le z \le |g|, 0 \le q \le |h|\},\end{aligned} \tag{25}$$

where g is the sub-state set containing states associated with an individual server (e.g., CPU utilization), and z corresponds to its index. Additionally, h signifies the sub-state set containing states associated with two servers (e.g., propagation time), and q denotes the index. Consequently, $\mathbb{S}$ is defined as:

$$\mathbb{S} = \{S_t = (\mathbb{F}_t(\mathcal{A}_i^j), \mathbb{G}_t(\mathcal{N}))|\mathcal{A}_i^j \in \mathcal{A}_i, t \in \mathbb{T}\}. \tag{26}$$

- *Action space* $\mathbb{A}$: In this work, scheduling involves the action of assigning the current task $\mathcal{A}_i^j$ to an individual server $\mathcal{N}_k$. Consequently, the definition of the action at time step t is as follows:

$$A_t = \mathcal{C}_i^j = \mathcal{N}_k. \tag{27}$$

Therefore, the action space $\mathbb{A}$ equals to the server set N:

$$\mathbb{A} = \mathcal{N}. \tag{28}$$

- *Reward function* $\mathbb{R}$: As outlined in Section III-B-4, the primary objective is to minimize the weighted cost model presented in (17). Thus, in time step t, the reward r_t can be defined as the negative value of (15) if the task can be successfully executed. However, if the task $\mathcal{A}_i^j$ fails to be executed on the scheduled server $\mathcal{C}_i^j$, a substantial negative value is introduced as a penalty. Formally, r_t is defined as:

$$r_t = \begin{cases} -J(\mathcal{C}_i^j) & succeed \\ penalty & fail, \end{cases} \tag{29}$$

IV. TF-DDRL: Distributed DRL Framework

The high-level architecture of the TF-DDRL framework is depicted in Fig. 2. The architecture comprises multiple Actors responsible for collecting data to create experience trajectories and a Learner that leverages the experience trajectories to learn a policy π. The architecture comprises multiple distributed Actors, which are responsible for interacting with the environment, collecting data, and generating experience trajectories by executing tasks based on their local policies. These experience trajectories are then sent to a Learner, whose role is to aggregate these trajectories, update the global policy π, and broadcast the updated policy back to the Actors to ensure consistent and improved decision-making across the system. Both the Actors and the Learner can be flexibly deployed on edge or cloud servers, depending on the system's requirements. For example, deployment on cloud servers may be preferred when higher computational capacity is needed, whereas edge servers may be more suitable when low latency or data locality is prioritized. The primary objective is to identify a policy π that maximizes the expected sum of future discounted rewards:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t \in T} \gamma_t r_t\right], \tag{30}$$

where π represents the policy, $\gamma \in [0, 1]$ is the discount factor, $r_t = r(s_t, a_t)$ denotes the reward at time t, s_t is the state at time t, s is the initial state s_0, and $a_t = \pi(a_t|s_t)$ is the action generated by following a specific policy π. Fig. 3 presents an overview of the TF-DDRL framework. In what follows, each component and communication process is discussed in detail.

A. Actor: Experience Trajectories Generation

Algorithm 1 describes how the Actor in the TF-DDRL framework generates experience trajectories. In order to improve the efficiency of sampling and the speed of convergence of TF-DDRL, PER is introduced to store the trajectory experiences of the Actor. At the beginning, the Actor updates its local policy μ to the most recent Learner policy π and initializes a PER buffer $\mathcal{P}$ to store the collected transitions. Before one trajectory, the Actor generates the initial state based on the information of the current task and server set. After that, based on the output a_t of the policy μ, the Actor schedules the current task to the corresponding server. Then, the reward r_t of the current action a_t is calculated based on (29), and the next state s_{t+1} is also generated based on the information of the next task and the server set. Afterward, the Actor calculates the importance measure m_t and stores the current transition $(s_t, a_t, r_t, \mu(a_t|s_t), s_{t+1})$ into $\mathcal{P}$ based on m_t. After n steps, the Actor sends the trajectory $\{s_1, a_1, r_1, \mu(a_1|s_1), \ldots, s_n, a_n, r_n, \mu(a_n|s_n), s_{n+1}\}$ to the Learner. The Learner then iteratively updates its policy π over a batch of trajectories gathered from different Actors. This framework

decouples data collection and learning, allowing more Actors to be added and distributed across multiple machines for efficient utilization of computing resources in edge and cloud IoT systems.

Algorithm 1: Actor: Experience Generation.

Input : the Actors's local policy μ; the Learner's policy π; the Learner's address $Learner$; max time step n;

while *True* **do**
  $\mu \leftarrow UpdateActorPolicy(\pi, Learner)$;
  $\mathcal{P} \leftarrow InitializePERBuffer()$;
  $servers \leftarrow GetServers()$;
  $task \leftarrow GetTask()$;
  $s_1 \leftarrow GenerateState(servers, task)$;
  for $t \leftarrow x$ *to* $x+n-1$ **do**
    $a_t \leftarrow \mu(s_t)$;
    $Schedule(task, a_t)$;
    $r_t \leftarrow GetReward()$;
    $servers \leftarrow GetServers()$;
    $task \leftarrow GetTask()$;
    $s_{t+1} \leftarrow GenerateState(servers, task)$;
    $e_t = (s_t, a_t, r_t, \mu(a_t|s_t), s_{t+1})$;
    $m_t = r_t + \gamma_t V(s_{t+1}) - V(s_t)$;
    Store transition e_t into $\mathcal{P}$ based on m_t;
  end for
  if $Length(\mathcal{P}) == n$ **then**
    $SubmitTrajectory(\mathcal{P}, Learner)$;
  end if
end while

B. Learner: Schedule Policy Update

However, it's worth noting that after a few updates, the Actor's strategy μ may fall behind the Learner's strategy π. To address the gap between the Actor's policy μ and the Learner's policy π, an off-policy correction method named V-trace [13] is introduced to rectify this discrepancy.

1) V-Trace Correction Method: The Learner in TF-DDRL maintains a state value function V based on the samples from the Actors. The purpose of the V-trace correction method is to provide an estimate of the current state value function V, called V-trace target $\hat{V}$. After n steps of interaction with the environment, the Actor collects a trajectory $(s_t, a_t, r_t, \mu(a_t|s_t), s_{t+1})_{t=x}^{t=x+n}$ following its policy μ. The n-steps V-trace target $\hat{V}(s_x)$ for state s_x is:

$$\hat{V}(s_x) = V(s_x) + \sum_{t=x}^{x+n-1} \gamma^{t-x} \left(\prod_{i=x}^{t-1} c_i \right) \delta_t V, \tag{31}$$

where $\delta_t V$ is the truncated Temporal Difference (TD) for V, and $\prod_{i=x}^{t-1} c_i$ measures the impact of $\delta_t V$ observed at time t on the update of the value function V at the previous time x. Specifically, $\delta_t V$ is defined as:

$$\delta_t V = \rho_t(r_t + \gamma_t V(s_{t+1})) - V(s_t)), \tag{32}$$

and c_i and ρ_t are truncated importance sampling weights,:

$$c_i = \min\left(\bar{c}, \frac{\pi(a_i|s_i)}{\mu(a_i|s_i)}\right), \tag{33}$$

$$\rho_t = \min\left(\bar{\rho}, \frac{\pi(a_t|s_t)}{\mu(a_t|s_t)}\right), \tag{34}$$

where $\bar{c}$ and $\bar{\rho}$ are the truncation constants with $\bar{c} \leq \bar{\rho}$. $\bar{c}$ affects the speed of convergence, while $\bar{\rho}$ affects the solution to which the value function V converges. Considering $\bar{\rho}$, the corresponding target policy $\pi_{\bar{\rho}}(a|x)$ is defines as:

$$\pi_{\bar{\rho}}(a|x) = \frac{min(\bar{\rho}\mu(a|x), \pi(a|x))}{\sum_{b \in A} min(\bar{\rho}\mu(b|x), \pi(b|x))} \tag{35}$$

2) Actor-Critic-Based Algorithm: The implementation of TF-DDRL follows the Actor-Critic architecture. TF-DDRL optimizes two DNNs, the actor (policy) network and the critic (value) network. The actor network focuses on acquiring a policy π to maximize the expected cumulative discounted reward $\mathbb{E}_\pi[\sum_{t \in T} \gamma_t r_t]$. Meanwhile, the critic network evaluates the current policy π by computing the TD error, which measures the difference between the current reward and the estimate of the value function V.

Algorithm 2 describes how the Learner in the TF-DDRL framework updates policies. The Learner first obtains the collected trajectories from all Actors. In order to improve the efficiency of sampling and the speed of convergence of the algorithm, trajectory experiences are sampled based on importance measure m_x. When updating the networks, the loss function of TF-DDRL is defined as follows:

$$loss_{total} = a_v * loss_{value} + a_p * loss_{policy} + a_e * loss_{entropy}, \tag{36}$$

where $loss_{value}$ is the loss function for value function, $loss_{policy}$ is the loss function for policy, $loss_{entropy}$ is the loss function for entropy bonus, and a_v, a_p, and a_e are the corresponding weights. Considering π_ϕ is the current policy parameterized by ϕ, V_θ is the value function parameterized by θ, and μ is the Actor's local policy, the value loss function $loss_{value}$ is defined as the L_2 loss between the current value V_θ and the V-trace target value $\hat{V}$:

$$loss_{value} = (\hat{V}(s_x) - V_\theta(s_x))^2, \tag{37}$$

where $\hat{V}(s_x)$ is from (31). Considering the objective function (30), the policy gradient can be presented as:

$$\nabla V^\pi(s) = \mathbb{E}_\pi[\nabla \log \pi(a_x|s_x) Q_\pi(s_x, a_x)], \tag{38}$$

where $Q_\pi(s_x, a_x)$ is the state-value of policy π at (s_x, a_x). In TF-DDRL, the truncated importance sampling weight ρ_x between the policy $\pi_{\bar{\rho}}$ and the Actor's local policy μ is employed to suppress the divergence. Also, we use $r_s + \gamma v_{s+1}$, named as the v-trace advantage, to estimate $Q_{\pi_{\bar{\rho}}}(a_x|s_x)$. Besides, state-dependent baseline $V_\theta(s_x)$ is subtracted from the v-trace advantage to reduce bias. Therefore, the policy loss function $loss_{policy}$ is defined as:

$$loss_{policy} = -\rho_x \log \pi_\phi(a_x|s_x)(r_x + \gamma v_{x+1} - V_\theta(s_x)), \tag{39}$$

where ρ_x is from (34). We also exploit the entropy $H(\pi_\phi)$ as a bonus to encourage exploration, with the loss function $loss_{entropy}$ defined as:

$$loss_{entropy} = -H(\pi_\phi) = \sum_a \pi_\phi(a_x|s_x) \log \pi_\phi(a_x|s_x) \tag{40}$$

Therefore, the value function parameter θ is updated in the direction of:

$$\Delta\theta = a_v * (\hat{V}(s_x) - V_\theta(s_x)) \nabla_\theta V_\theta(s_x), \tag{41}$$

and the policy parameter ϕ is updated through policy gradient:

$$\Delta\phi = a_p \rho_x \nabla_\phi \log \pi_\phi(a_x|s_x)(r_x + \gamma v_{s+1} - V_\theta(s_x)) - a_e \nabla_\phi \sum_a \pi_\phi(a_x|s_x) \log \pi_\phi(a_x|s_x). \tag{42}$$

Algorithm 2: Learner: Policy Update.

Input : current policy π_ϕ; value function V_θ; update epoch X; buffer size N; value function loss coefficient a_v; policy objective function loss coefficient a_c; entropy bonus loss coefficient a_e; the Actors set $Actors$

while *True* **do**
  $\mathcal{D} \leftarrow InitializeBuffer()$;
  for *actor in Actors* **do**
    $\mathcal{D}.append(ReceiveTrajectory(actor))$;
  end for
  for *trajectory in* $\mathcal{D}$ **do**
    for $x \leftarrow 1$ *to* X **do**
      Sample experience $e_x = (s_x, a_x, r_x, \mu(a_x|s_x), s_{x+1}) \sim \mathcal{M}(x) = m_x^\alpha / \sum_i m_i^\alpha$;
      $w_x = (N\mathcal{M}(x))^{-\beta} / max_i w_i$;
      $\hat{V}(s_x) \leftarrow V_\theta(s_x) + \sum_{t=x}^{x+n-1} \gamma^{t-x} (\prod_{i=x}^{t-1} c_i) \delta_t V_\theta$;
      $m_x \leftarrow |\delta_t|$;
      $loss_{value} \leftarrow (\hat{V}(s_x) - V_\theta(s_x))^2$;
      $loss_{policy} \leftarrow -\rho_x log\pi_\phi(a_x|s_x)(r_x + \gamma v_{x+1} - V_\theta(s_x))$;
      $loss_{entropy} \leftarrow \sum_a \pi_\phi(a_x|s_x) log\pi_\phi(a_x|s_x)$;
      $loss_{total} \leftarrow a_v * loss_{value} + a_p * loss_{policy} + a_e * loss_{entropy}$;
      $\Delta\theta \leftarrow \Delta\theta + w_x a_v (\hat{V}(s_x) - V_\theta(s_x)) \nabla_\theta V_\theta(s_x)$;
      $\Delta\phi \leftarrow \Delta\phi + w_x (a_p \rho_x \nabla_\phi log\pi_\phi(a_x|s_x)(r_x + \gamma v_{s+1} - V_\theta(s_x)) - a_e \nabla_\phi \sum_a \pi_\phi(a_x|s_x) log\pi_\phi(a_x|s_x))$;
    end for
    update θ and ϕ by Adam optimizer;
  end for
  $BroadcastPolicy(Actors, \pi_\phi)$;
end while

C. Prioritized Experience Replay

The Learner in TF-DDRL relies on the experience trajectories collected from Actors to update the parameters. However, in dynamic edge and cloud environments, the experience changes over time, resulting in significant gaps between samples. Each sample can contribute to different improvements to the model. To enhance sampling efficiency, expedite convergence, and enable the model to quickly adapt to changes by focusing on the most pertinent experiences during the training phase, PER [16] is introduced in both the data collection phase and the model update phase.

As presented in Algorithm 1, during the data collection phase, the Actor assigns importance measure m_t to each experience sample when storing it in the buffer. Since the TD error reflects the difference between the model's estimated value of the current state and the next state, and when this difference is significant, it indicates that the experience sample provides valuable information for updating the current policy. Therefore, TF-DDRL uses the TD error as a metric to measure the importance of samples, defined as:

$$m_t = r_t + \gamma_t V(s_{t+1}) - V(s_t) + \epsilon, \tag{43}$$

where ϵ is a tiny positive number from 0 to 1, in case the experience is not sampled when the TD error is 0. As presented in Algorithm 2, when the Learner samples experience e_x from the trajectory, the sampling probability $\mathcal{M}(x)$ is calculated as follows:

$$e_x \sim \mathcal{M}(x) = \frac{m_x^\alpha}{\sum_i m_i^\alpha} \tag{44}$$

where α determines the degree of priority, and $\alpha = 0$ corresponds to the uniform case (i.e., each experience has the same probability of being sampled).

However, when experiences are given priority, they have different probabilities of being sampled, which will introduce bias in the update of the value network, thus changing the direction of the convergence of the value network. In order to correct this error, an importance-sampling weight is added to each empirical sample, calculated as follows:

$$w_x = \left(\frac{1}{N\mathcal{M}(x)}\right)^\beta * (\max_i w_i)^{-1}, \tag{45}$$

where N is the number of experience samples in the PER buffer, β is a hyperparameter within 0 and 1 that will gradually increase and finally settle at 1, and $(\max_i w_i)^{-1}$ is to normalize the weight to improve stability. The purpose of using the importance-sampling weight is to strike a balance between prioritizing samples to learn important experiences and reducing the potential bias. As β continues to rise to 1, the bias gradually decreases, and the learning process gradually reduces the impact of prioritization, ensuring a more stable and unbiased learning process. This helps prevent the model from becoming too sensitive to specific experiences, encouraging a more robust and accurate learning process.

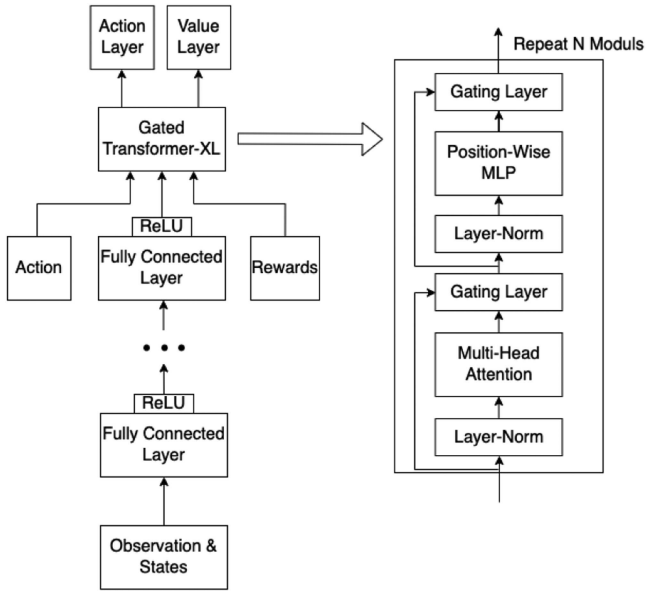


Fig. 4. The network architecture of TF-DDRL framework.

D. Gated Transformer-XL

Due to the heterogeneity and dynamics of edge and cloud environments, TF-DDRL uses the Gated Transformer-XL [14] to allow the model to better capture long-term dependencies and global relationships between states. The network architecture of TF-DDRL is shown in Fig. 4.

In the Transformer layer of TF-DDRL, the Multi-Head Attention block applies the attention mechanism to different linear mappings (heads) of the input and concatenates them together, allowing the model to focus on different parts of the input sequence simultaneously, which helps to capture the relationships between different features in the input. The Position-wise Multi-Layer Perceptron (MLP) block is used to perform independent nonlinear transformations of features at each position, enhancing the model's ability to capture complex patterns and relationships at different positions in the input sequence, providing a more expressive representation for downstream processing. The Gating Layer is used to weight the features of each position when passing through different blocks. The model can control the flow of input information by learning the

appropriate weights, making it more suitable for specific tasks and data distribution.

While the inclusion of Transformers in TF-DDRL can enhance the model's ability to capture long-term dependencies in scheduling tasks, it also introduces additional computational overhead due to its multi-head attention mechanism. However, by leveraging the distributed Actor-Learner architecture, TF-DDRL distributes the computational burden across multiple servers, thus mitigating the impact on individual nodes. This design choice allows the framework to maintain scalability and efficiency in dynamic IoT environments, despite the added complexity. Additionally, the use of PER further optimizes sampling efficiency, reducing the exploration costs associated with training.

V. Performance Evaluation

This section introduces the experiment configuration, hyperparameters of the TF-DDRL, and the performance study.

A. Experiment Setup

We discuss the specification of our practical edge-cloud environment, details of employed real IoT applications, and baseline techniques.

1) Practical Experiment Environment: To reflect the heterogeneous computing environments, a practical experiment environment, containing IoT devices, edge servers, and cloud servers, is established. Besides, to build a multi-cloud computing environment, we used three instances from the Nectar Cloud infrastructure (All AMD EPYC with 2 cores @2.0 GHz, 8 GB RAM; 4cores @2.0 GHz, 16 GB RAM; 8 cores @2.0 GHz, 32 GB RAM), one instance from AWS Cloud (Intel Xeon with 1 core @2.4 GHz, 1 GB RAM), and one instance from Microsoft Azure Cloud (Intel Xeon with 1 core @2.3 GHz, 1 GB RAM).

In the edge computing environment, we used one RPi 3B (Pi OS, Broadcom BCM2837 with 4 cores @1.2 GHz, 1 GB RAM), one Macbook Pro (macOS, M1 Pro with 8 cores, 16 GB RAM), and one Dell laptop (Linux, Intel Core i7 with 8 cores @2.3 GHz, 16 GB RAM). Also, as IoT devices, we have used webcams, IP cameras, and docker containers that stream pre-recorded video files.

Moreover, we used the Victorian Default Offer[1] (i.e., 0.2871 AUD/kWh) in Australia as the electricity price, and the official price of AWS[2] and Microsoft Azure[3] cloud servers (i.e., 0.1296 AUD/hour for m6a.large, 0.2592 AUD/hour for m6a.xlarge, 0.5184 AUD/hour for m6a.2xlarge, 0.0174 AUD/hour for t2.micro, and 0.0156 AUD/hour for B1s) to calculate the monetary cost in the experiment. In our environment, the servers exhibit the following average latency and bandwidth (data rate): the latency between the IoT device and the Nectar cloud server ranges from 6-12 ms, with a bandwidth between 14-20 MB/s; between the IoT device and the AWS cloud server, the latency ranges from 15-25 ms, with a bandwidth between 15-22 MB/s; between the IoT device and the Microsoft Azure cloud server, the latency ranges from 7-15 ms, with a bandwidth between 15-21 MB/s; the latency between the IoT device and the edge servers ranges from 1-6 ms, with a bandwidth between 130-140 MB/s. The energy consumed to execute applications on servers is monitored using the eco2AI library [47]. Moreover, the transmission power $W^{tr}(\mathcal{C}_i^j)$ of servers is obtained similar to [40], [41], and $W^{tr}(\mathcal{C}_i^j)$ is set between 0.75-1 W for edge servers and between 3-5 W for cloud servers. However, these parameters can be adjusted.

Furthermore, in (16), w_1, w_2, and w_3 are set to 0.33, indicating that the importance of response time, energy consumption, and monetary cost are considered equal.

2) Sample IoT Applications: To evaluate TF-DDRL's performance, we utilized four types of IoT applications, featuring real-time and/or non-real-time capabilities. Real-time functionality allows applications to process live streams, while non-real-time functionality facilitates the processing of pre-recorded video files. These applications, adhering to a sensor-actuator architecture, are dynamically distributed across the heterogeneous IoT devices in both edge and cloud environments. Each application can operate on multiple devices simultaneously, creating a realistic and diverse application scheduling environment. Additionally, all applications offer an adjustable parameter known as the *application label*, which determines the resolution of the video. The applications are detailed below:

- *Face Detection* [48]: Identifies human faces in real-time, marking them with squares in the video. This application is implemented using the OpenCV.[4]
- *Color Tracking* [48]: Traces colors in a video stream in real-time. Users have the flexibility to dynamically configure target colors using the application's GUI. This application is developed using OpenCV[4].
- *Face And Eye Detection* [48]: Alongside identifying human faces in real-time, it detects human eyes. This application is developed using OpenCV[4].
- *Video OCR* [49]: Retrieves textual content from pre-recorded video and presents it to the user. It is designed to automatically filter keyframes for efficient processing. This application is developed using Google's Tesseract-OCR Engine.[5]

3) Baseline Techniques: To evaluate the performance of TF-DDRL, we implemented five additional DRL techniques, including centralized and distributed, as outlined below:

- *IMPALA* [13]: It is a distributed DRL technique and is designed for large-scale environments. TF-DDRL is based on the architecture of IMPALA to enable high scalability in highly distributed environments.
- *ApeX-DQN* [17]: It is an improved DRL technique based on DQN that introduces a distributed learning architecture, adopted by Wen et al. [33] for scheduling problems.
- *A3C* [18]: It is one of the most adapted techniques in the distributed DRL field for scheduling problems. It has been used by many works in the current literature, including [34], [35], [36], [37], and [38]. It combines the Actor-Critic method with the concept of concurrent execution. We extend this technique to solve the proposed optimization problem in the heterogeneous edge and cloud computing environment.
- *D3QN-RNN:* Many works ([22], [24], [26], [27], [30], and [31]) use DQN-based DRL techniques. We extend the foundation of DQN, incorporating the Dueling architecture [19] to decompose the Q values into state and advantage values for a more accurate estimation of the relative value of actions. Also, we introduce Double DQN [20], employing two independent neural networks to estimate target Q-values to address the overestimation during training. Moreover, RNN is used in this technique.
- *SAC* [21]: It is a centralized DRL technique and is used by Zheng et al. [28]. It combines the Actor-Critic method with

[1] https://www.esc.vic.gov.au/electricity-and-gas/prices-tariffs-and-benchmarks/victorian-default-offer

[2] https://aws.amazon.com/pricing

[3] https://azure.microsoft.com/pricing

[4] https://github.com/opencv/opencv

[5] https://github.com/tesseract-ocr/tesseract

TABLE III
THE HYPERPARAMETERS SETTING FOR TF-DDRL

TF-DDRL Hyperparameter	Value
Fully Connected Layers	3
Hidden Layer Units	[256, 256, 128]
Activation Function	ReLU
Learning Rate lr	0.001
Discount Factor γ	0.99
Transformer Unit Number	2
Transformer Head Number	4
Transformer Head Dimension	32
Transformer Position-wise MLP Dimension	32
Optimization Method	Adam

TABLE IV
HYPERPARAMETERS OF BASELINE TECHNIQUES

Hyperparameters	ApeX-DQN	A3C	D3QN-RNN	SAC
Fully Connected Layers	3	3	3	3
Hidden Layer Units	[256,256,128]	[256,256,128]	[256,256,128]	[256,256,128]
Activation Function	ReLU	TanH	ReLU	ReLU
Learning Rate	0.001	0.001	0.001	0.0001
Discount Factor	0.99	0.9	0.99	0.99

entropy regularization, encouraging exploration, and enhancing the stability of learning. It is extended to address our problem within the heterogeneous edge and cloud computing environment.

B. Technique Hyperparameters

The network architecture of TF-DDRL is depicted in Fig. 4. In our implementation, we used three fully connected layers, followed by two Gated Transformer-XL-based attention layers, and then two additional fully connected layers for generating action logits and the value function. Furthermore, we performed a grid search to fine-tune the hyperparameters. Accordingly, we set the learning rate (lr) to 0.001 and the discount factor (γ) to 0.99. Also, the $\bar{c}$ and $\bar{\rho}$, governing the V-trace performance, are both set to 1 for optimal results. Table III provides a summary of the hyperparameter settings. Moreover, we conducted hyperparameter tuning for the baseline techniques to ensure a fair assessment of their performance, as presented in Table IV.

C. Performance Study

The results of our extensive experiments are shown below.

1) PER and Transformer Analysis: This experiment studies the performance of TF-DDRL compared to native IMPALA. We employ the four applications detailed in Section V-A-2 for training. Due to the page limit, the results are provided exclusively for weighted costs.

Fig. 5 shows the outcome of TF-DDRL under various model configurations. Without the use of both PER and Transformer, the native IMPALA requires approximately 90 iterations to converge to the optimal solution discovered in the experiment. The convergence speed slightly improves when only PER is employed. However, with the exclusive employment of the Transformer, TF-DDRL demonstrates a significant acceleration in convergence speed, reaching the experiment's optimal solution in around 50 iterations. When both PER and Transformer are used concurrently, TF-DDRL converges in approximately 40 iterations.

These results clearly highlight the advantages of integrating both PER and Transformer within TF-DDRL. While incorporating the Transformer could potentially introduce challenges such as instability and slower convergence due to increased model complexity, the V-trace correction mechanism ensures stable learning during the distributed training process. Additionally, PER further accelerates learning by prioritizing important experiences. This synergistic combination enables TF-DDRL to find better scheduling solutions more efficiently compared to native IMPALA.

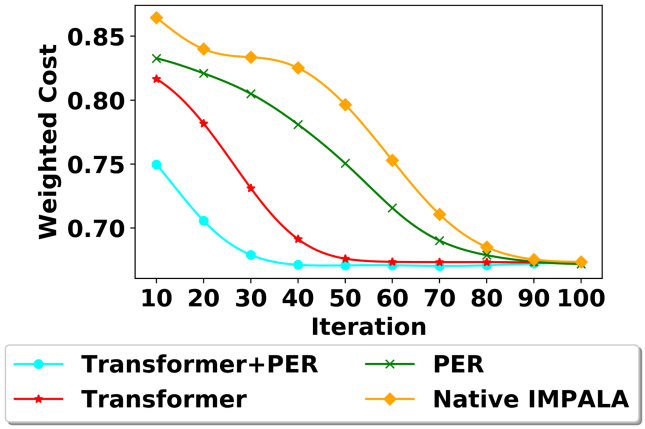


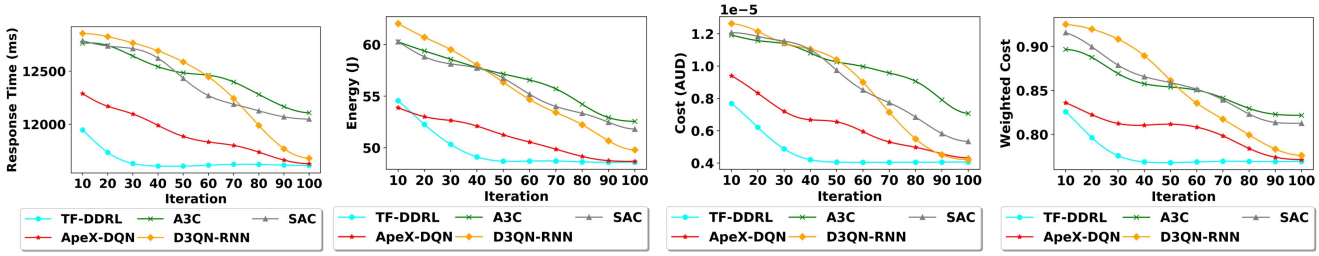
Fig. 5. PER and Transformer analysis.

2) Cost Versus Policy Update Analysis: This experiment analyzes the performance of TF-DDRL in various iterations during policy updates. For training purposes, we utilize four applications as detailed in Section V-A-2, configuring the resolution as 480. The results, showing the policy cost versus updating iteration, are presented in Fig. 6.

As shown in Fig. 6, the optimization costs of all techniques decrease with the increasing number of iterations in different scenarios. However, under different optimization objectives, TF-DDRL shows a faster convergence compared to other techniques. It converges to the best scheduling solution discovered during training in approximately 40 iterations. ApeX-DQN exhibits a slower convergence speed than TF-DDRL but eventually converges to the optimal scheduling solution in 90 iterations. D3QN-RNN converges to the best scheduling solution under monetary cost optimization Fig. 6(c). Although the costs of A3C and SAC decrease continuously during training, neither of them converges to the optimal solution within 100 iterations.

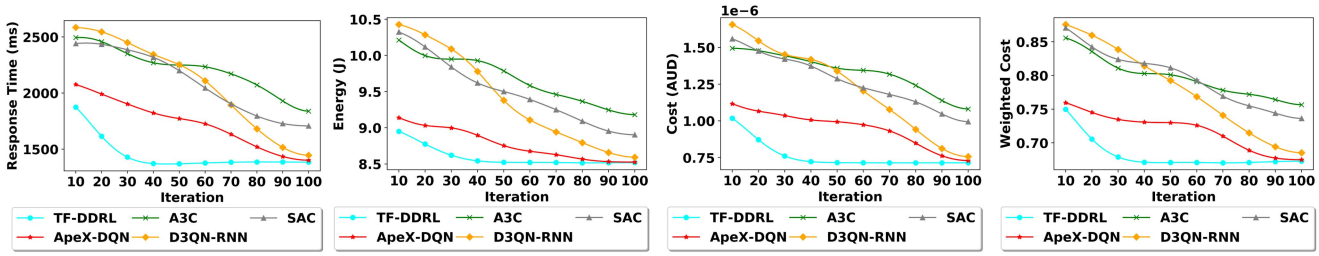
During evaluation, the resolution is adjusted to 240, altering the IoT application's demands for computing resources compared to the training phase. The results, showing the optimization cost versus the policy update for various algorithms, are presented in Fig. 7. It is obvious that similar to the results obtained during the training phase, compared with other techniques, TF-DDRL demonstrates better performance in response time, energy consumption, monetary cost, and weighted cost in the evaluation phase. Also, after 100 iterations of updates for all baseline techniques, none of them can achieve results superior to TF-DDRL. This indicates that TF-DDRL not only converges faster, with significantly less time compared to other techniques but also provides better scheduling results. Except for the ApeX-DQN technique, A3C, D3QN-RNN, and SAC do not converge to the optimal scheduling solution in 100 iterations. Overall, compared to ApeX-DQN results, which is the only baseline technique that converges to the optimal scheduling solution found in the evaluation phase across all optimization objectives, TF-DDRL achieves average performance gains of 60% , 51% , 56% , and 58% in response time, energy consumption, monetary cost, and weighted cost, respectively.

These performance advantages of TF-DDRL can be attributed to several key technical designs. The Transformer, with its multi-head



(a) Response time (b) Energy Consumption (c) Monetary Cost (d) Weighted cost

Fig. 6. Cost versus policy update analysis - training phase.



(a) Response time (b) Energy Consumption (c) Monetary Cost (d) Weighted cost

Fig. 7. Cost versus policy update analysis - evaluation phase.

attention mechanism and position-wise MLP layers, effectively captures complex dependencies between state features and provides strong non-linear modeling capabilities, enabling better generalization to different resource configurations. The PER mechanism enables more efficient experience sampling by prioritizing informative experiences based on TD errors, significantly reducing exploration costs compared to uniform sampling methods used in baseline approaches. Additionally, the combination of distributed experience sharing and V-trace off-policy correction ensures efficient utilization of collected experiences while maintaining training stability, addressing the limitations of techniques like A3C that rely on local experiences. These design elements collectively contribute to TF-DDRL's superior convergence speed and scheduling performance in dynamic edge and cloud environments.

3) Scalability Analysis: This experiment investigates the impact of different numbers of servers on the scheduling technique for IoT applications. The number of available servers directly impacts the complexity of IoT application scheduling problems, as a higher number of servers leads to a larger action space. To evaluate the scalability performance of TF-DDRL, the experiment uses varying numbers of servers (e.g., 5, 10, 15, 20, 25, 30). Also, other settings are consistent with Section V-C-2. Due to space constraints and the fact that the results for response time, energy consumption, and monetary cost follow the same patterns as weighted costs, only the results for weighted costs are presented.

Fig. 8 shows the weighted cost optimization results obtained by various techniques after 100 iterations, considering the growth of candidate servers. As the number of servers increases, TF-DDRL consistently outperforms other techniques, converging more rapidly towards superior solutions. This shows that as the system scales up, TF-DDRL demonstrates superior scalability, enabling it to make more effective application scheduling decisions in fewer iteration cycles. In the baseline techniques, ApeX-DQN outperforms other techniques, although weighted costs eventually continue to increase with the growth of available servers.

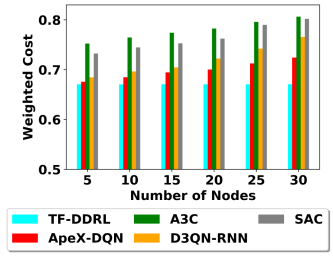


Fig. 8. Scalability analysis.

The superior scalability of TF-DDRL can be attributed to its architectural advantages in handling large-scale environments. The Transformer's self-attention mechanism efficiently processes the increasing state space by dynamically focusing on relevant server features, while its position-wise MLP provides the necessary modeling capacity for complex server relationships. Moreover, the distributed experience collection combined with PER ensures efficient exploration of the expanded action space, as it prioritizes experiences that are most informative for learning optimal scheduling policies. These design elements allow TF-DDRL to maintain its performance even as the server count increases, whereas baseline methods struggle with the exponentially growing state-action space.

4) Greenhouse Gas Emission Analysis: This experiment examines the impact of various scheduling techniques based on Greenhouse Gas Emission (GHE). We specifically analyze electricity generation patterns in Australia,[6] the US,[7] and Germany,[8] considering the associated GHE of various sources involved in

[6] https://www.energy.gov.au/data/electricity-generation
[7] https://www.eia.gov/tools/faqs
[8] https://www.umweltbundesamt.de/themen/co2-emissionen-pro-kilowattstunde-strom-stiegen-in

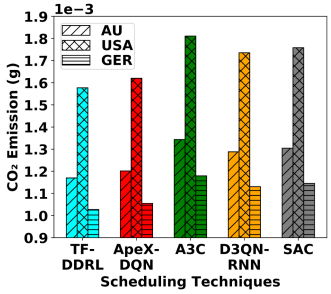


Fig. 9. GHE analysis.

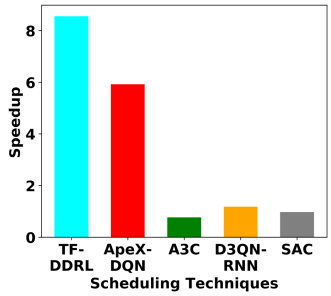


Fig. 10. Speedup analysis.

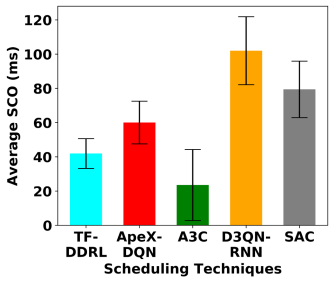


Fig. 11. SCO analysis.

electricity production.[9] The total GHE is defined as the sum of GHE from the production of electricity from each source [50], shown below:

$$GHE = EC * \sum_{i \in Sources} (U_i * P_i), \tag{46}$$

where EC represents the total electricity consumed, U_i represents the amount of greenhouse gas emitted per unit of electricity produced using source i, and P_i represents the proportion of the source i in producing electricity.

Fig. 9 presents the GHE associated with different scheduling techniques based on electricity generation in different countries. Notably, TF-DDRL exhibits the lowest GHE, while A3C has more GHE compared to other techniques. Also, the GHE based on the US power generation pattern is substantially higher than that of Australia and Germany. This discrepancy is due to the prevalence of fossil sources, including coal and natural gas, in the US electricity generation pattern. The experiment results show that TF-DDRL can effectively reduce GHE, which can contribute to collective efforts to address climate change, alleviate the impacts of global warming, and foster a healthier and more sustainable natural environment.

TF-DDRL achieves lower GHE through several key technical advantages in its scheduling decisions. The Transformer architecture enables more precise modeling of the relationship between server power consumption patterns and application characteristics, leading to more energy-efficient task scheduling. Additionally, the PER mechanism helps identify and prioritize experiences that lead to energy-saving scheduling strategies. The distributed learning framework further allows TF-DDRL to explore and learn from a diverse range of energy-efficient scheduling patterns. These capabilities result in more intelligent resource utilization and reduced energy waste, thereby effectively decreasing GHE across different power generation patterns.

5) Speedup Analysis: With the similar experimental configuration outlined in Section V-C-2, we explore the speedup performance of various techniques. We define the reference time, denoted as $Time_r$, as the time required for the weighted cost of TF-DDRL with an Actor to reach a value of 0.76. Designating 0.76 as the reference weighted cost is motivated by the fact that this particular value serves as the smallest weighted cost that can be obtained by all baseline techniques. Additionally, we define $Time_t$ as the time required by each technique to attain the reference weighted cost. So, the speedup SPU for each technique is defined as follows:

$$SPU = \frac{Time_r}{Time_t}. \tag{47}$$

The speedup results for all techniques are illustrated in Fig. 10. The results demonstrate that TF-DDRL outperforms A3C, D3QN-RNN, and SAC by 7 to 11 times, and it is over 40% faster than ApeX-DQN. This significant speedup advantage can be attributed to several key designs of TF-DDRL. The Transformer's parallelism enables efficient processing of state information. The PER mechanism further accelerates learning by focusing computational resources on the most informative experiences. Moreover, TF-DDRL's V-trace correction method effectively addresses the policy lag between the Actor and Learner, providing more stable and efficient training. These architectural advantages collectively enable TF-DDRL to achieve faster learning and better adaptation to dynamic edge and cloud computing environments.

6) Scheduling Overhead (SCO) Analysis: This experiment investigates the SCO of each technique. We use the same environment settings in Section V-C-2. For each technique, we run 100 iterations, each containing four IoT applications. Also, the average SCO is defined as $Time_a = \frac{Time_o}{100}$, where $Time_o$ denotes the total overhead of the technique to schedule the IoT applications.

Fig. 11 illustrates the $Time_a$ within the 95% confidence interval for various techniques during the scheduling of IoT applications. The scheduling overhead of TF-DDRL is lower compared to ApeX-DQN, D3QN-RNN, and SAC, but higher than A3C. The higher overhead is expected due to the employment of Transformer layers and PER mechanism. However, this trade-off between overhead and performance is well justified by TF-DDRL's significantly better scheduling decisions and faster convergence. Thus, in heterogeneous edge and cloud computing environments, TF-DDRL proves to be more efficient in scheduling IoT applications.

VI. Conclusions and Future Work

In this paper, we proposed a distributed DRL technique, named TF-DDRL, designed to solve DAG-based IoT application scheduling in highly heterogeneous and dynamic edge and cloud computing environments. We formulated the IoT application scheduling problem as an optimization problem and then transformed it into an MDP model, aiming to minimize response time, energy consumption, monetary cost, and weighted cost. We proposed the TF-DDRL,

[9] https://world-nuclear.org/information-library/energy-and-the-environment/carbon-dioxide-emissions-from-electricity.aspx

which follows Actor-Critic architecture, incorporating PER and Transformer techniques to decrease exploration costs and enhance convergence speed. TF-DDRL allows multiple parallel and scalable Actors to work simultaneously and share experience trajectories with the Learner, enabling more effective and efficient learning. Also, we used the V-trace off-policy correction method to solve discrepancies between Learner and Actor policies. As demonstrated by extensive experiments, in highly stochastic and heterogeneous computing environments, TF-DDRL possesses better scalability and adaptability, compared to its counterparts. The results indicate that TF-DDRL outperforms other DRL-based approaches, demonstrating performance improvements of up to 60% , 51% , 56% , and 58% in terms of response time, energy consumption, monetary cost, and weighted cost, respectively.

As part of future work, we will consider more aspects of the optimization problem, including system load balancing. Also, we plan to develop a resource management framework based on TF-DDRL for edge and cloud environments, allowing users to customize and evaluate applications and scheduling policies in dynamically heterogeneous environments. Furthermore, we intend to explore the integration of recent transformer-based DRL advancements, such as [51], particularly for scenarios involving offline data-driven IoT application scheduling. Additionally, we plan to investigate the design and deployment of multi-Learner architectures within the TF-DDRL framework to improve scalability and adaptability in large-scale edge and cloud environments.

REFERENCES

[1] M. Goudarzi, M. Palaniswami, and R. Buyya, "Scheduling IoT applications in edge and fog computing environments: A taxonomy and future directions," *ACM Comput. Surv.*, vol. 55, no. 7, pp. 1–41, 2022.

[2] F. Al-Doghman, N. Moustafa, I. Khalil, Z. Tari, and A. Zomaya, "AI-enabled secure microservices in edge computing: Opportunities and challenges," *IEEE Trans. Serv. Comput.*, vol. 16, no. 2, pp. 1485–1504, Mar./Apr. 2023.

[3] X. Su, L. An, Z. Cheng, and Y. Weng, "Cloud–edge collaboration-based bi-level optimal scheduling for intelligent healthcare systems," *Future Gener. Comput. Syst.*, vol. 141, pp. 28–39, 2023.

[4] W. Wen et al., "Health monitoring and diagnosis for geo-distributed edge ecosystem in smart city," *IEEE Internet Things J.*, vol. 10, no. 21, pp. 18571–18578, Nov. 2023.

[5] X. Dai et al., "Task co-offloading for D2D-assisted mobile edge computing in industrial Internet of Things," *IEEE Trans. Ind. Informat.*, vol. 19, no. 1, pp. 480–490, Jan. 2023.

[6] Z. Wang, M. Goudarzi, M. Gong, and R. Buyya, "Deep reinforcement learning-based scheduling for optimizing system load and response time in edge and fog computing environments," *Future Gener. Comput. Syst.*, vol. 152, pp. 55–69, 2023.

[7] S. Pallewatta, V. Kostakos, and R. Buyya, "Placement of microservices-based IoT applications in fog computing: A taxonomy and future directions," *ACM Comput. Surv.*, vol. 55, 2023, Art. no. 321.

[8] H. Ma, R. Li, X. Zhang, Z. Zhou, and X. Chen, "Reliability-aware online scheduling for DNN inference tasks in mobile edge computing," *IEEE Internet Things J.*, vol. 10, no. 13, pp. 11453–11464, Jul. 2023.

[9] X. Zhou, S. Ge, P. Liu, and T. Qiu, "DAG-based dependent tasks offloading in MEC-enabled IoT with soft cooperation," *IEEE Trans. Mobile Comput.*, vol. 23, no. 6, pp. 6908–6920, Jun. 2023.

[10] X. Wang et al., "Deep reinforcement learning: A survey," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 35, no. 4, pp. 5064–5078, Apr. 2024.

[11] W. Chen, X. Qiu, T. Cai, H.-N. Dai, Z. Zheng, and Y. Zhang, "Deep reinforcement learning for Internet of Things: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 3, pp. 1659–1692, Third Quarter 2021.

[12] V. Mnih et al., "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2016, pp. 1928–1937.

[13] L. Espeholt et al., "IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2018, pp. 1407–1416.

[14] E. Parisotto et al., "Stabilizing transformers for reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2020, pp. 7487–7498.

[15] A. Vaswani, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6000–6010.

[16] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2015, *arXiv:1511.05952*.

[17] D. Horgan et al., "Distributed prioritized experience replay," 2018, *arXiv:1803.00933*.

[18] V. Mnih et al., "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2016, pp. 1928–1937.

[19] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2016, pp. 1995–2003.

[20] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.

[21] 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. Int. Conf. Mach. Learn.*, PMLR, 2018, pp. 1861–1870.

[22] M. Bansal, I. Chana, and S. Clarke, "UrbanEnQoSPlace: A deep reinforcement learning model for service placement of real-time smart city IoT applications," *IEEE Trans. Serv. Comput.*, vol. 16, no. 4, pp. 3043–3060, Jul./Aug. 2023.

[23] L. T. Hoang, C. T. Nguyen, and A. T. Pham, "Deep reinforcement learning-based online resource management for UAV-assisted edge computing with dual connectivity," *IEEE/ACM Trans. Netw.*, vol. 31, no. 6, pp. 2761–2776, Dec. 2023.

[24] L. Huang, X. Feng, C. Zhang, L. Qian, and Y. Wu, "Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing," *Digit. Commun. Netw.*, vol. 5, no. 1, pp. 10–17, 2019.

[25] L. Zhao et al., "MESON: A mobility-aware dependent task offloading scheme for urban vehicular edge computing," *IEEE Trans. Mobile Comput.*, vol. 23, no. 5, pp. 4259–4272, May 2024.

[26] L.-T. Hsieh, H. Liu, Y. Guo, and R. Gazda, "Deep reinforcement learning-based task assignment for cooperative mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 23, no. 4, pp. 3156–3171, Apr. 2024.

[27] Y. Fan, J. Ge, S. Zhang, J. Wu, and B. Luo, "Decentralized scheduling for concurrent tasks in mobile edge computing via deep reinforcement learning," *IEEE Trans. Mobile Comput.*, vol. 23, no. 4, pp. 2765–2779, Apr. 2024.

[28] Y. Zheng, H. Zhou, R. Chen, K. Jiang, and Y. Cao, "Sac-based computation offloading and resource allocation in vehicular edge computing," in *Proc. IEEE Conf. Comput. Commun. Workshops*, 2022, pp. 1–6.

[29] J. Wang, L. Zhao, J. Liu, and N. Kato, "Smart resource allocation for mobile edge computing: A deep reinforcement learning approach," *IEEE Trans. Emerg. Topics Comput.*, vol. 9, no. 3, pp. 1529–1541, Third Quarter 2021.

[30] X. Xiong, K. Zheng, L. Lei, and L. Hou, "Resource allocation based on deep reinforcement learning in IoT edge computing," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1133–1146, Jun. 2020.

[31] X. Jie, T. Liu, H. Gao, C. Cao, P. Wang, and W. Tong, "A DQN-based approach for online service placement in mobile edge computing," in *Proc. 16th EAI Int. Conf. Collaborative Comput.: Netw., Appl. Worksharing*, Springer, 2021, pp. 169–183.

[32] X. Chen, S. Hu, C. Yu, Z. Chen, and G. Min, "Real-time offloading for dependent and parallel tasks in cloud-edge environments using deep reinforcement learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 35, no. 3, pp. 391–404, Mar. 2024.

[33] S. Wen, R. Han, C. H. Liu, and L. Y. Chen, "Fast DRL-based scheduler configuration tuning for reducing tail latency in edge-cloud jobs," *J. Cloud Comput.*, vol. 12, no. 1, pp. 1–32, 2023.

[34] Z. Wang, M. Li, L. Zhao, H. Zhou, and N. Wang, "A3C-based computation offloading and service caching in cloud-edge computing networks," in *Proc. IEEE Conf. Comput. Commun. Workshops*, 2022, pp. 1–2.

[35] R. Garaali, C. Chaieb, W. Ajib, and M. Afif, "Learning-based task offloading for mobile edge computing," in *Proc. IEEE Int. Conf. Commun.*, 2022, pp. 1659–1664.

[36] Y. Ju et al., "NOMA-assisted secure offloading for vehicular edge computing networks with asynchronous deep reinforcement learning," *IEEE Trans. Intell. Transp. Syst.*, vol. 25, no. 3, pp. 2627–2640, Mar. 2024.

[37] B. Sellami, A. Hakiri, and S. B. Yahia, "Deep reinforcement learning for energy-aware task offloading in join SDN-blockchain 5G massive IoT edge network," *Future Gener. Comput. Syst.*, vol. 137, pp. 363–379, 2022.
[38] S. Chen, J. Chen, Y. Miao, Q. Wang, and C. Zhao, "Deep reinforcement learning-based cloud-edge collaborative mobile computation offloading in industrial networks," *IEEE Trans. Signal Inf. Process. Netw.*, vol. 8, pp. 364–375, 2022.
[39] J. Hao et al., "Exploration in deep reinforcement learning: From single-agent to multiagent domain," *IEEE Trans. Neural Netw. Learn. Syst.*, ol. 35, no. 7, pp. 8762–8782, Jul. 2024.
[40] E. Meskar, T. D. Todd, D. Zhao, and G. Karakostas, "Energy efficient offloading for competing users on a shared communication channel," in *Proc. IEEE Int. Conf. Commun.*, 2015, pp. 3192–3197.
[41] X. Long, J. Wu, and L. Chen, "Energy-efficient offloading in mobile edge computing with edge-cloud collaboration," in *Proc. Int. Conf. Algorithms Architectures Parallel Process.*, 2018, pp. 460–475.
[42] 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.
[43] Y. Ding, K. Li, C. Liu, Z. Tang, and K. Li, "Budget-constrained service allocation optimization for mobile edge computing," *IEEE Trans. Serv. Comput.*, 2021.
[44] N. H. Tran, W. Bao, A. Zomaya, M. N. Nguyen, and C. S. Hong, "Federated learning over wireless networks: Optimization model design and analysis," in *Proc. IEEE INFOCOM*, 2019, pp. 1387–1395.
[45] J. Ji, K. Zhu, and L. Cai, "Trajectory and communication design for cache-enabled UAVs in cellular networks: A deep reinforcement learning approach," *IEEE Trans. Mobile Comput.*, vol. 16, no. 1, pp. 147–161, Jan./Feb. 2023.
[46] M. Goudarzi, M. A. Rodriguez, M. Sarvi, and R. Buyya, "μ-DDRL: A QoS-aware distributed deep reinforcement learning technique for service offloading in fog computing environments," *IEEE Trans. Serv. Comput.*, vol. 17, no. 1, pp. 47–59, Jan./Feb. 2024.
[47] S. A. Budennyy et al., "Eco2AI: Carbon emissions tracking of machine learning models as the first step towards sustainable AI," *Doklady Math.*, vol. 106, pp. S118–S128, 2022.
[48] M. Goudarzi, Q. Deng, and R. Buyya, "Resource management in edge and fog computing using FogBus2 framework," 2021, *arXiv:2108.00591*.
[49] Q. Deng, M. Goudarzi, and R. Buyya, "FogBus2: A lightweight and distributed container-based framework for integration of iot-enabled systems with edge and cloud computing," in *Proc. Int. Workshop Big Data Emergent Distrib. Environments*, 2021, pp. 1–8.
[50] N. Scarlat, M. Prussi, and M. Padella, "Quantification of the carbon intensity of electricity produced and used in Europe," *Appl. Energy*, vol. 305, 2022, Art. no. 117901.
[51] L. Chen et al., "Decision transformer: Reinforcement learning via sequence modeling," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, pp. 15084–15097.

Zhiyu Wang is working toward the PhD degree with the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, University of Melbourne. His research focuses on edge/fog computing, the Internet of Things (IoT), distributed systems, and artificial intelligence. He is particularly interested in applying AI techniques to optimize resource management in dynamic edge, fog, and cloud computing environments.

Mohammad Goudarzi received the PhD degree from the University of Melbourne's Department of Computing and Information Systems, in 2022. He is an assistant professor with the Department of Software Systems and Cybersecurity, Monash University. His research is centered on developing advanced solutions for large-scale distributed systems, with a particular focus on the Internet of Things (IoT), cloud/edge computing, applied machine learning, and applied security. He was a senior research associate with UNSW Sydney and Cybersecurity CRC, where he collaborated with Cisco on a joint project. In 2024, he was selected as one of the top 200 young computer science and mathematics scientists by the prestigious Heidelberg Laureate Forum (HLF). His achievements have also earned him several notable awards, including Oracle's Cloud Architect of the Year Award 2022, the IEEE TCCLD Outstanding PhD Thesis Award 2022, the IEEE TCSC Outstanding PhD Dissertation Award 2022, and the IEEE Outstanding Service Award 2021.

Rajkumar Buyya (Fellow) is a Redmond Barry distinguished professor and director with the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, University of Melbourne, Australia. He has authored more than 850 publications and seven textbooks including "Mastering Cloud Computing" published by McGraw Hill, China Machine Press, and Morgan Kaufmann for Indian, Chinese, and international markets, respectively. He is one of the highly cited authors in computer science and software engineering worldwide (h-index=170, g-index=374, 155, 100+ citations).