







Multi-Agent Reinforcement Learning for Task Offloading in Crowd-Edge Computing

Su Yao , Mu Wang, Ju Ren , Senior Member, IEEE, Tianyu Xia , Weiqiang Wang, Ke Xu , Fellow, IEEE, Mingwei Xu , Senior Member, IEEE, and Hongke Zhang , Fellow, IEEE

Abstract—The Crowd-edge (CE) computing paradigm facilitates the utilization of the computational resources through simultaneously relying the edge computing and the collaboration among various mobile devices (MDs). Most existing works, focusing on offloading tasks from device to edge servers by centralized solutions, are unable to distribute tasks to massive MDs in CE. Meanwhile, designing a decentralized task offloading solution enabling task subscribers to individually make offloading decisions can be challenging given the randomness of crowd resource provisioning and limited knowledge of global status variations. In this paper, we propose a decentralized crowd-edge task offloading solution that enables users to optimally offload tasks to the CE in a distributed manner. Specifically, we formulate the corresponding problem as a stochastic optimization with partially observable status. By observing network and process delays at the crowd side, we further reform the optimization forms and provide a novel approximation policy, enabling users to optimize their offloading strategy based on local observations without interaction with each other. We then solve this task offloading problem by developing a Mixed Multi-Agent Proxy Policy Optimization algorithm (mixed MAPPO). Extensive testing, including numerical and system-level simulations, was conducted to validate the performance of the proposed algorithm in terms of task delay (including the processing delay and transmission delay), load rate, and resource utilization.

Index Terms—Task offloading, edge computing, multi-agent reinforcement learning, distributed optimization.

Received 4 August 2024; revised 3 December 2024; accepted 2 January 2025. Date of publication 21 January 2025; date of current version 3 September 2025. This work was supported in part by the National Natural Science Foundation of China under Grant 62394322, Grant U22B2031, Grant 62471066, Grant 62394323, Grant 62472240, Grant 62132009, Grant 61932016, Grant 62132011, and Grant 92167204, in part by the Science Fund for Creative Research Groups of the National Natural Science Foundation of China under Grant 62221003, in part by the Ant Group through CCF Ant Innovative Research Program under Grant CCF-AFSGRF20210023, in part by the Beijing National Research Center for Information Science and Technology under Grant BNR2025RC01010 and Grant BNR2023TD03006, and in part by the project of State Key Laboratory of Networking and Switching Technology under Grant NST20250203. Recommended for acceptance by X. Yu. (Su Yao and Mu Wang contributed equally to this work.) (Corresponding author: Mu Wang.)

Su Yao, Ju Ren, Ke Xu, and Mingwei Xu are with the Beijing National Research Center for Information Science and Technology (BNRist), Tsinghua University, Beijing 100190, China (e-mail: yaosu@tsinghua.edu.cn; renju@tsinghua.edu.cn; xuke@tsinghua.edu.cn; xumw@tsinghua.edu.cn).

Mu Wang is with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China (e-mail: muwang@bupt.edu.cn).

Tianyu Xia and Weiqiang Wang are with Ant Group, Hangzhou 310000, China (e-mail: xty368914@antgroup.com; weiqiang.www@antgroup.com).

Hongke Zhang is with the School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing 100044, China (e-mail: hkzhang@bjtu.edu.cn).

Digital Object Identifier 10.1109/TMC.2025.3531793

I. INTRODUCTION

THE current upsurge of mobile applications such as Artificial Intelligence (AI), autopilot, Industrial Internet-of-things (IIoT), and blockchains are reshaping our lifestyles and gaining tremendous attention from both the industry and academia. These applications, however, are computationally intensive and delay-sensitive, which makes them unlikely to be processed by hand-held devices. An intuitive way to compensate resource shortage of user devices is by offloading the computation tasks of these applications to Cloud centers [1], [2]. The extra delay introduced by remote Cloud may impair users' quality of experience, especially the multimedia and AI-based services that require delivering a considerable volume of data for processing. The Cloud's drawbacks can be overcome by implementing powerful computation units in proximity to users, which is known as edge computing [3], [4]. However, high-performance edge computing may require deploying many edge servers to constantly meet the high demand of various applications, which is economically undesirable.

Recent advances in distributed [5], [6] and collaborative learning [7], [8] have enabled numerous geographically distributed computation units to process data cooperatively. Consequently, researchers have become increasingly interested in exploiting the resources available in various mobile devices (MDs) [9], [11]. This has led to the emergence of a new computation paradigm known as crowd-edge computing [12]. In particular, there is a growing trend to integrate computing and networking [10] to enable a distributed computing paradigm that supports various AI services. This approach, known as computation-integrated networking, relies heavily on efficient collaboration between crowd and edge resources. Fig. 1 illustrates how crowd-edge (CE) computing incorporates MDs with elastic resource provision, allowing for the simultaneous mitigation of resource shortage, backbone inefficiency, and delay issues. In addition, CE benefits from edge servers, which help to stabilize resource provision and meet quality-of-service demands. Despite the promising potential of CE, enabling high-performance task offloading to the crowd-edge is a non-trivial task and remains problematic. Moreover, the heterogeneous and complex computational structure in CE (i.e., the capacity varies in different MDs) makes the designation of offloading logic much more complex than the conventional edge computing paradigm. Another problem is that the unpredictable nature of resource provision at MDs yields the fact that it is unlikely to

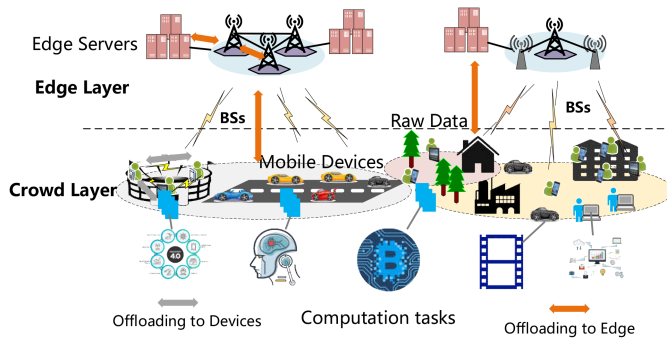


Fig. 1. Task offloading in crowd-edge system.

derive the optimal offloading simply based on the observations of the instant system status. Several studies [24], [25] have attempted to formulate the task offloading problem in CE as a non-convex optimization and to demonstrate its NP-hardness. However, due to the dynamic nature of CE systems, which are subject to changing user demand and unpredictable network conditions, the assumption that the modeling system is static in these studies is infeasible. Hence, the results of these problems are subject to sub-optimality due to the application of heuristic algorithms.

Reinforcement Learning (RL) [18], [19], methods tailored for the optimal controls in dynamic systems, open new opportunities to crowd-edge task offloading [13], [14]. RL-based methods formulate the task offloading problem as a Markov Decision Process (MDP), aiming to maximize the long-term cumulative utility. The learning agent interacts with the environment by observing the variations of the environment after making decisions. During the interactions, the agent estimates the long-term reward from decision-making and builds the optimal offloading strategy based on the observations. The generated policies sequentially decide the optimal task offloading at each episode to approximate the long-term optimum. Current RL-based task offloading in crowd-edge [27], [28], [29] mainly uses a central learning framework relying on the knowledge of global environment dynamics and assuming users can decide the offloading scheme synchronously. Yet crowd-edge users, in reality, can only observe the dynamic of the partial environment and make decisions asynchronously. Namely, a distributed task offloading scheme enables each participant to independently decide which computation units to offload tasks are more applicable to crowd-edge task offloading. However, the design of distributed task offloading schemes is not well studied, and several challenges need to be tackled:

- 1) *Partial observability*: Existing task offloading approaches often operate under the assumption that the decision-maker possesses complete visibility into the system's dynamics. In practice, these individuals are constrained by limited knowledge and lack access to comprehensive data regarding system fluctuations, such as network link capacity and computation unit resource utilization. The complexity arises when striving to make decisions that aim for a global optimum while working with only partial information, presenting a notable challenge.

- 2) *Distributed decision-making*: In a crowd-edge environment, users are often both numerous and geographically dispersed. This characteristic renders centralized task offloading schemes economically unattractive. However, crafting an effective distributed task offloading scheme that allows users to make task offloading decisions by their own is no simple feat. Achieving a consensus among diverse agents to reach a global optimum necessitates information exchange, which, in turn, introduces additional communication overhead to the system.
- 3) *Network Congestion*: Many current solutions tend to treat network delay as either a random or static parameter. In reality, network performance is intricately intertwined with the offloading process itself, as the growing influx of arrival requests can significantly impact the transmission links, potentially causing congestion which in turn impairs the offloading performance.

We propose a novel multi-agent reinforcement learning-based task offloading scheme to address these challenges. We first model the network and computation resource dynamic and formulate the task offloading problem as stochastic optimization. We further discuss how to transform this stochastic optimization into a partial observable Markov decision process (POMDP). We then propose a Mixed Multi-Agent Proxy Policy Optimization algorithm (mixed MAPPO) algorithm to perform the task offloading when both the network and computation status are unknown to the users. Extensive tests have been conducted to validate the performance of our proposed task offloading in terms of delay and resource usage. The contributions of this paper are multi-folded:

- 1) We have defined the crowd-edge task offloading problem as a stochastic optimization problem that can be formed as a POMDP with the overarching goal of jointly optimizing long-term processing and transmission delays. The unknown status of edge computing resources and network conditions make such a problem nontrivial to be solved by conventional methods.
- 2) By observing the forms of the formulated POMDP, we first estimate the end-to-end delay of transmission and edge processing delay based on the transformation of the penalty forms of the original problem. Then, we apply the reinforcement learning algorithm of MAPPO to solve the estimated forms of the problem. A major difference between our proposed methods and current MARL is we do not rely on the learning model to deal with the partial observability, which may introduce extra computation and errors.
- 3) We also conduct extensive numerical and simulation tests to compare the proposed method with several state-of-the-art solutions. Simulation results show the feasibility and advantages of our proposed algorithm.

The remainder of the paper is organized as follows: The background and related works are reviewed in Section II. Section III shows the system model. Section IV discusses how to formulate the task offloading for crowd-edge. Section V introduces the designation of mixed MAPPO. Section VI evaluates

mixed MAPPO via both the numerical and prototype-level tests and Section VII concludes the paper.

II. BACKGROUND AND RELATED WORK

Thus far, many studies have been dedicated to the designation of task offloading in crowd-edge scenarios. Hong et al. in [23] exploit the computational resources from not only the Cloud and edge but also the devices of end users. The corresponding offloading problem is formulated as a multi-task partitioning optimization problem and solved by an online offloading mechanism. He et al. in [24] formulated the task offloading problem as a mixed-integer problem and decoupled it into two subproblems. The first subproblem aims to minimize resource usage for a given device-to-device (D2D) communication pair, while the second subproblem aims to maximize system capacity. Wen et al. aimed to improve energy efficiency by offloading tasks via D2D links, enhancing overall performance as well [25]. Xing et al. [26] minimize the computation latency by D2D since the tasks can be offloaded to nearby mobile equipment. A suboptimal task offloading algorithm by relaxing the original mix-integer problem to convex optimization is then proposed.

Typically, the aforementioned offloading strategies formulate the corresponding problem as a nonconvex optimization, whose solution can only reach suboptimal. Besides, these studies with deterministic optimization overlook that the resource provision and task arrivals are time-varying. The gap between the real world and system modeling can degrade system performance. One promising approach to tackle this issue is to use RL methods. Yi et al. in [27] apply deep reinforcement learning (DRL) to offload computational-intensive tasks. Min et al. in [28] also propose an RL-based solution to offload the task of IoT devices to edge devices. In [29], Chen et al. allow users to offload computation tasks to multiple base stations. They formulate the offloading problem as a MDP and apply a double deep Q-network (DQN) method to learn the optimal offloading strategy.

Despite the effectiveness of these RL-based solutions, implementing such a centralized learning paradigm for task offloading in Crowd-edge is nontrivial, because of the difficulty in timely capturing the dynamic of global status. Besides, the fast-increasing number of mobile devices can quickly exhaust the capacity of the central coordinator. Multi-agent reinforcement learning (MARL) tailored for distributed decision-making is a possible solution to tackle these problems in Crowd-edge task offloading. Its performance has been proved in much literature dedicated to task offloading in mobile edge computing. For instance, Zhang et al. [30] consider the coexistence of multiple Cloud centers. Considering the self-interest of each Cloud center and resource competition between them, a MARL-based distributed task offloading scheme is provided, enabling Cloud centers to collaboratively determine the task offloading by observing the others' decisions. In [31], Liu et al. aim to minimize the average task completion time for task offloading. Considering the users are numerous and make decisions independently, an RL-based algorithm based on a counterfactual MARL approach is given.

In [32], Nguyen et al. integrate mobile edge computing with blockchain and propose a MARL-based task offloading and

TABLE I
TASK OFFLOADING DEDICATED TO DEVICE-ASSISTED COMPUTATION SCENARIOS

	Crowd Adoption	Stochastic	Long-term Optimality	Decentralized
FCFI [23], [24], [25], [26]	✓	×	×	×
[27], [28], DSARL [29]	✓	✓	✓	×
[30], [31], [32], MAT [33]	×	✓	✓	✓
Ours	✓	✓	✓	✓

block mining scheme to offer task offloading solutions with security enhancement. Wang et al. in [33] apply the MARL algorithm to optimize the unmanned aerial vehicle trajectory programming which ensures the connectivity between the edge computing system and user equipment (UEs). Based on the trajectories, they further gave a low-complexity optimization method for offloading tasks from UEs to the edge servers. In our previous work [35], we consider a blockchain assist framework for the task offloading in Cloud-edge-device scenarios. Specifically, blockchain workers reach a consensus by solving the task offloading problem.

Unlike the above methods [30], [31] that only consider the task offloading to Cloud or edge servers, our solutions are designed for crowd-edge with heterogeneous resource capacities and complex system environments. Different from our work [35] which focuses on incorporating the blockchain to foster a truthful task offloading environment, this work proposes a novel MARL-based algorithm to achieve a better performance on the task offloading. Instead of directly optimizing the cumulative reward as in conventional RL-based methods, we develop a novel probabilistic control-based RL algorithm by evaluating the probability of a state being optimum. The comparison of the existing works is shown in Table I.

III. SYSTEM MODELING

This section introduces the task offloading procedure in crowd-edge and the system model, including the communication and computation models. The notations used in the model are given in Table II.

A. Task Offloading in Crowd-Edge

Fig. 1 shows the task offloading in crowd-edge. The crowd-edge can be abstracted as a two-layer computation architecture: The edge layer consists of the edge computing servers that are in proximity to the users yet have limited computation capacity; The Device layer consists of nearby MDs with enormous redundant idle resources to exploit but unstable in provision. Users decide to offload the generated task to one of the above layers by their interest. For example, the task's raw data can be offloaded to the Cloud center (as the red arrow lines in the figure) by traversing the backbone network and also offloaded to local edge computing servers or MDs.

B. Communication Model

We make the following assumptions for the data delivery in crowd-edge: (1) Edge computing servers directly connect the

TABLE II
NOTIFICATIONS

Notation	Definition
\mathbb{V}, \mathbb{E}	set of nodes and links in system
\mathcal{T}	time slots set of the crowd-edge model
$V_u, u = e, m$	computation units in crowd-edge
$l(u, v)$	links between nodes u and v
c_l	link capacity of l
$F_u(t)$	available computation resources of u at t
$s_i(t)$	computation resources required for processing task i
o_i	size of data needs to be offloaded by i
$Q_u(t)$	task resource requesting backlogs at t
$\mathcal{P}_{i,u}(t)$	total processing time of task i
$x_i(t)$	data transmission rate for task i during t
Δt	time length of slot t
$V_l(t)$	data backlogs of l at t
$E_{i,t}$	execution delay of i
\bar{x}_i	average transmission rate
R_u	unit resource usage fee of u
$\Pr(a)$	probability of a
$E_{ij}(t)$	link set used by provider-client pair (i, j)
V	penalty parameter
\mathcal{N}	set of the agents
\mathcal{S}	state space
\mathcal{A}	action space
\mathcal{O}	observation space

base station for low latency task offloading; (2) Client can use base station to relay the offloading request to MDs within the same cell; (3) We denote the universe of the time slots in system by $\mathcal{T} = \{1, 2, \dots, T\}$.

The network can be represented as an undirected graph $G(\mathbb{V}, \mathbb{E})$, where \mathbb{V} denotes the set of nodes and \mathbb{E} denotes the set of links. Assume $V_u, u = e, m$ are the computation units handling the offloaded tasks. For any of the link $l \in \mathbb{E}$, let c_l denote its capacity.

We use $\mathcal{T}_i(t) := s_i(t), b_i$ to represent task i at time t , where $s_i(t)$ is the required computation resources and o_i is the size of the raw data to be processed. Let $x_i(t)$ denote the data rate for delivering task i at time t , and $o_i = \sum_{t=t_0}^{t'} x_i(t) \Delta t$, where t_0 is the time slot when data transmission begins, and t' is the time slot when data processing begins. Δt denotes the length of the time slot. The average transmission rate is \bar{x}_i , and we have $o_i = (t' - t_0) \Delta t \bar{x}_i$. Since $(t' - t_0) \Delta t$ is equal to o_i / \bar{x}_i , a higher transmission rate yields a lower waiting delay before processing a task. Since the link the capacity¹ of l is c_l , the overall data rate over the l subject to the capacity constraint,

$$\sum_{i \in H_l(t)} x_i(t) \leq c_l \quad (1)$$

$$V_l(t+1) = V_l(t) + \sum_{i \in H_l(t)} x_i(t) - c_l \quad (2)$$

where $H_l(t)$ is the set of tasks using l during t . For the long-term perspective of inequality (1),

$$\lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=1}^T \sum_{i \in H_l(t)} x_i(t) \leq c_l \quad (3)$$

¹Our model can be also extended to the condition of link capacity is time-varying, by rephrasing the c_l to $c_l(t)$.

Virtual queue value $V_l(t)$ should be *stable* [37]. Stability in the context of our virtual queue optimization signifies a delicate balance between the rate of arrivals and the service rate, preventing the queue from experiencing unchecked growth. Specifically, for any stable queue with the length denoted as V_l , its arrival rate $\sum_{i \in H_l(t)} x_i(t)$ is no larger than the service rate in average. Ensuring the stability of the virtual queue guarantees that computation nodes and transmission links will not become inundated by incoming tasks over the long term, preserving the system's efficiency and performance.

C. Computation Model

For computing node $u \in V_u, u = e, m$, its available resources is $F_u(t)$. We measure u capacity of the processing task by its CPU frequency (i.e., GHz). The resources provided by MDs can be time-varying due to the randomness of human behaviors and MDs only support the offloading with idle resources. We assume

$$F_v(t) \gg F_w(t), v \in V_e, w \in V_m$$

This is because, in the real world, edge computing servers are more capable than MDs in terms of task offloading. Task i is executed when the target computation unit receives the raw data for the task. To measure the processing delay of task i associated with computation unit u , we define the backlog of tasks awaiting processing by $Q_u(t)$. Let the set of tasks allocated to u during t is denoted by $A_u(t)$, and the processing rate is denoted by $F_u(t)$. Hence, the dynamic of $Q_u(t)$ can be described as follows:

$$Q_u(t+1) = Q_u(t) + \sum_{i \in A_u(t)} s_i(t) - F_u(t) \quad (4)$$

By Little's law, the total processing delay of task i is the waiting delay and serving time,

$$\mathcal{P}_{i,u}(t) = \frac{Q_u(t) + s_i(t)}{F_u(t)} \quad (5)$$

IV. PROBLEM FORMULATION

A. Stochastic Optimization Problem of Task Offloading

We define the task execution time by:

$$E_{i,t} = \mathcal{P}_{i,u}(t) + \frac{o_i}{\bar{x}_i} \quad (6)$$

For computation node u , if the resource usage is R_u , the cost of processing task i at u can be defined as $R_u s_i(t)$. According to execution delay and processing cost, we define the cost of offloading i to u :

$$\alpha E_{i,t} + \beta R_u s_i(t) \quad (7)$$

where $\alpha, \beta \in [0, 1]$ are the weights. Specifically, the dimension of α is 1/seconds.

To minimize the overall task executions delay and resource usage jointly, the task offloading problem (P1) can be formulated as follows:

$$\text{Minimize } \sum_{t=1}^T \sum_{i \in G(t)} (\alpha E_{i,t} + \beta \mathbf{w}_i(t) \mathbf{R}_u^T s_i(t)) \quad (8)$$

Subject to $V_l(t)$ stable, $\forall l \in \mathbb{E}$ (9)

By (8), **P1**'s purpose is to minimize the overall cost of the system. In (8), $G(t)$ is the set of the task generated during t . $\mathbf{w}_i(t)$ is one-hot vector with $|V_e \cup V_m|$ dimensions and its components $w_{iu}(t)$:

$$w_{iu}(t) = \begin{cases} = 1, & \text{offload } i \text{ to } u \text{ at } t \\ = 0, & \text{otherwise} \end{cases} \quad (10)$$

$\mathbf{R}_u^T := \{R_u\}_{u \in V_e \cup V_m}$. The constraint (9) data rate should not exceed the network capacity. Instead of directly optimizing the original problem with queue stability constraints (8) and (9), we employ a transformation technique outlined in Lemma 4.6 and Theorem 4.8 from [37]. This transformation leads us to an unconstrained optimization problem (12), which theoretically approximates the optimal solution for (8) and (9). The process of transforming (8) and (9) is analogous to the derivations presented in [37] as (4.48)(4.49). To provide context, we initially consider the drift-plus-penalty form of (8) and (9): $V \sum_{i \in G(t)} (\alpha E_{i,t} + \beta \mathbf{w}_i(t) \mathbf{R}_u^T s_i(t)) + \sum_{l \in \mathbb{E}} \Delta \mathcal{V}_l(t)$ of (8) and (9). Here, $\Delta \mathcal{V}_l(t) := \frac{1}{2}(V_l^2(t+1) - V_l^2(t))$. This form is an upper bound as follows:

$$B + V \sum_{i \in G(t)} (\alpha E_{i,t} + \beta \mathbf{w}_i(t) \mathbf{R}_u^T s_i(t)) + \sum_{l \in \mathbb{E}} V_l(t) \left(\sum_{i \in H_l(t)} x_i(t) - c_l \right) \quad (11)$$

Disregarding the constant term B, we obtain the expression (12). For an in-depth derivation process, please refer to Section 4.5 in [37]. Hence, we solve the following problem at each t :

$$\begin{aligned} \text{Minimize } & V \sum_{i \in G(t)} (\alpha E_{i,t} + \beta \mathbf{w}_i(t) \mathbf{R}_u^T s_i(t)) \\ & + \sum_{l \in \mathbb{E}} V_l(t) \left(\sum_{i \in H_l(t)} x_i(t) - c_l \right) \end{aligned} \quad (12)$$

Let V denote the penalty which is nonnegative. Based on the optimization (12), we can reformulate the above problem as the following POMDP:

\mathcal{N} : the set of users in crowd-edge. Let the agents be the crowd-edge users who generated the tasks and make the offloading decisions. Accordingly, we rephrase the first term of (12) to $\sum_{i \in \mathcal{N}} (\alpha E_{i,t} + \beta R_u s_i(t))$ and second term to $\sum_{i \in H_l(t)} (x_i(t) - c_l)$.

\mathcal{S} : the state space. We define $\mathcal{S} = \{\mathcal{Q}, \mathcal{L}, \mathcal{F}\}$, where \mathcal{Q} is the set of $Q_u(t)$, \mathcal{L} is the value set of $V_l(t)$ for all links, \mathcal{F} the set of $F_u(t)$, $u \in V_e \cup V_m$.

\mathcal{A} : the action space. Clients decide which offload the task to which computation unit. To ease the description,² we assume each user generates one task at each t . Actions of client corresponding to task i at t can be denoted $(\mathbf{w}_i(t), x_i(t))$.

²We can also easily extend this assumption to the case of one user with multiple tasks by rephrasing $\mathbf{w}_i(t)$ to a matrix where each row is corresponding to the offloading decision of a task, and $x_i(t)$ to a vector $\mathbf{x}(t)$ with the component of $x_i(t)$

$r(\mathbf{w}(t), \mathbf{x}(t))$: instant reward. We define the $r(\mathbf{w}(t), \mathbf{x}(t))$ by the cost function of $\mathbf{w}(t)$ and $\mathbf{x}(t)$,

$$\begin{aligned} r(\mathbf{w}(t), \mathbf{x}(t)) \\ = V \sum_{i \in \mathcal{N}} (\alpha E_{i,t}(\mathbf{w}_i(t), x_i(t)) + \beta \mathbf{w}_i(t) \mathbf{R}_u^T s_i(t)) \\ + \sum_{l \in \mathbb{E}} V_l(t) \left(\sum_{i \in H_l(t)} x_i(t) - c_l \right) \end{aligned} \quad (13)$$

where

$$E_{i,t}(\mathbf{w}_i(t), x_i(t)) := \mathbf{w}_i(t) \mathbf{P}_i^T(t) + \frac{o_i - \sum_{v=t'_i}^{t-1} x_i(v)}{x_i(t)}$$

Let $\mathbf{P}_i(t) := \{\mathcal{P}_{i,u}(t)\}_{u \in V_e \cup V_m}$, and $\mathbf{P}_i^T(t)$ is the transpose of $\mathbf{P}_i(t)$, t'_i the time slot start to deliver i .

\mathcal{O} the observations \mathbf{o}_i of i . Considering each client is difficult to have the full knowledge of system dynamics. According to the formulated problem, two types of information are required when performing the optimization: (1) network status, including the dynamic of the available bandwidth of each link; (2) computation resource status, including the dynamic of the available resources at each computation unit. In practice, this information can be challenging to observe. For example, network operators may not be willing to reveal their topology information to the end users and edge computation operators since they can be in a conflict of interest. In such a context, the users can not measure the end-to-end delay of the data transmission. Namely, the network status is partially observed. The mobile device carriers are also unwilling to share their computation resource status due to privacy issues.

Next, we discuss how to solve crowd-edge task offloading problem under two different cases: (1) when network status is partially observed; (2) when Both the network and computation status are partially observed.

V. TASK OFFLOADING IN CROWD-EDGE

A. Distributed Stochastic Optimization for Partially Observed Network Status

Solving the formulated problem P1 remains the following challenges:

(1) In practice, network resources are maintained by different parties and are partially observable to the task coordinators. For example, $V_l(t)$ of each l is hard to get for the subscriber of i . Normally, clients can only observe end-to-end delay $D_i(t)$ whose approximation

$$D_i(t) \approx \sum_{l \in U_i(t)} \frac{V_l(t)}{c_l} \quad (14)$$

where $U_i(t)$ is the links used by i .

Due to the link being concurrently used by multiple requests, **P1** is hard to solve because the constraints are highly coupled distributed. Recall that the **P1** can be equally derived by solving the (13). The first term is the summation of task offloading

utilities. Since

$$\min_{i \in \mathcal{N}} \sum (g(\mathbf{w}_i(t), x_i(t))) = \sum_{i \in \mathcal{N}} \min(g(\mathbf{w}_i(t), x_i(t)))$$

for the function

$$g(\mathbf{w}_i(t), x_i(t)) := \alpha E_{i,t}(\mathbf{w}_i(t), x_i(t)) + \beta \mathbf{w}_i(t) \mathbf{R}_u^T s_i(t)$$

Thus, the first terms of (7) is decouple in terms of the $(\mathbf{w}_i(t), x_i(t))$. However, tasks over the same link are still coupled by the second term which indicates that optimizing their actions of data rate $x_i(t)$ should negotiate with each other to ensure the $V_l(t)$ is stable. To solve **P1** distributedly, we consider the optimal condition of (13)

$$\nabla_{(\mathbf{w}(t), \mathbf{x}(t))} r(\mathbf{w}(t), \mathbf{x}(t)) = 0 \quad (15)$$

where $(\nabla_{\mathbf{w}(t), \mathbf{x}(t)})$ denotes the subgradient of the function of $(\mathbf{w}(t), \mathbf{x}(t))$. Since the $\nabla_{\mathbf{w}(t), \mathbf{x}(t)} c_l$ in (15) equals to 0. The optimal condition (15) can be simplified to

$$\nabla_{\mathbf{w}(t), \mathbf{x}(t)} \left(V \sum_{i \in \mathcal{N}} g(\mathbf{w}_i(t), x_i(t)) + \sum_{l \in U_i(t)} V_l(t) x_i(t) \right) = 0 \quad (16)$$

And the (16) is also decoupled in terms of the task i . Thus, we can minimize the (12) distributedly by the following iterations:

For each task i :

$$\mathbf{w}_i(t+1) = \arg \max_{w_i} g(\mathbf{w}, \mathbf{x}(t)) \quad (17)$$

$$x_i(t+1) = \arg \max_{x_i} g(\mathbf{w}(t), \mathbf{x}) \quad (18)$$

Each link updates its $V_l(t)$ by:

$$V_l(t+1) = V_l(t) + (C(t) - \sum_{i \in H_l(t)} x_i(t)) \quad (19)$$

For each computation node u , update the $\mathcal{Q}_u(t)$ by

$$\mathcal{Q}_u(t+1) = \mathcal{Q}_u(t) + \sum_i w_{iu}(t) s_i(t) - F_u(t) \quad (20)$$

In the above iterations, the solution $(\mathbf{w}_i(t), x_i(t))$ for each i can be independently solved without coordinating with other tasks i by only observing the $V_l(t)$ of network links and $\mathcal{Q}_u(t)$ of computation nodes. However, recall that parameters $V_l(t)$ can not be directly observed, given the network typologies are transparent to clients. Consider the end-to-end delay $D_i(t)$ approximates $\sum_{l \in U_i(t)} (V_l(t)/c_l)$ by (14). Dividing the $g(\mathbf{w}(t), \mathbf{x}(t))$ by V , we have:

$$f(\mathbf{w}_i(t), x_i(t)) = V g(\mathbf{w}_i(t), x_i(t)) + \sum_{l \in U_i(t)} V_l(t) x_i(t) \quad (21)$$

We set the penalty V to a vector $\mathbf{V}_i := \{c_l\}_{l \in U_i(t)}$, and replace $g(\mathbf{w}_i(t), x_i(t))$ by

$$\begin{aligned} f'(\mathbf{w}_i(t), x_i(t)) &= g(\mathbf{w}_i(t), x_i(t)) + \sum_{l \in U_i(t)} \frac{V_l(t)}{c_l} x_i(t) \\ &\approx f(\mathbf{w}_i(t), x_i(t)) + D_i(t) x_i(t) \end{aligned} \quad (22)$$

Algorithm 1: Stochastic Optimization for Crowd-Edge Task Offloading.

Input: $\mathbf{w}, \mathbf{x}, t = 0$;
 initial queue length: $\mathcal{Q}_u(t), \overline{\mathcal{Q}}_m(t)$;
 link bandwidth c_l ;
 $\pi_{\omega^*}(\mathbf{a}|\mathbf{o})$;
 1 **while** $t \leq T$ **do**
 2 **foreach** *Agent* $i \in \{1, \dots, N\}$ **do**
 3 Estimate the $D_i(t)$;
 4 Calculate the $\hat{\mathbf{w}}_i(t), \hat{x}_i(t)$ by (22)(23);
 5 $\mathbf{w}_i(t), x_i(t) = \max\{\beta_t(\mathbf{s}_t, \mathbf{w}'_i(t), x'_i(t), \beta_t(\mathbf{s}_t, \hat{\mathbf{w}}'_i(t), \hat{x}'_i(t)))\}$
 6 **end**
 7 **end**
 8 **Computation Nodes:**
 9 Update the $\mathcal{Q}_u(t+1)$;
 10 **final** ;

Then, iterations (17)(18) can be reformed as:

$$\mathbf{w}_i(t+1) = \arg \max_{w_i} f'(\mathbf{w}, \mathbf{x}(t)) \quad (23)$$

$$x_i(t+1) = \arg \max_{x_i} f'(\mathbf{w}(t), \mathbf{x}) \quad (24)$$

The pseudo code of the above iterations is given in Algorithm 1.

B. Mixed MAPPO for Task Offloading Scheme

Efficiently offloading tasks by solving (22) and (23) on time can be challenging, primarily due to delays and asynchronicity in collecting status information from various sources. Therefore, we introduce a practical solution using a mixed MAPPO algorithm to address problem **P1**. This approach empowers each agent to autonomously explore optimal offloading options at time "t" by comparing actions derived from probabilistic control reinforcement learning and estimated stochastic optimization. Mixed MAPPO is a multi-agent variant of Proximal Policy Optimization (PPO) [36], which falls under policy gradient methods. PPO is known for its unique characteristic of employing a surrogate objective for policy updates, featuring a clipping term. This clipping mechanism ensures that the updated policy doesn't deviate excessively from the old one, preventing overly aggressive adjustments. In our proposed mixed MAPPO, we utilize an actor-critic network, where the policy is collectively evaluated by multiple critic networks based on historical data and feedback rewards. The actor network then generates actions by averaging the output values obtained from the multiple critic networks. The workflow of the proposed mixed MAPPO is shown in Fig. 2.

Remark: PPO is purpose-built to enhance sample efficiency, enabling agents to acquire effective policies with reduced interactions with the environment. This characteristic is particularly valuable in the context of crowd-edge computing, where collecting time-consuming network and resource status data can be a bottleneck. MAPPO boasts exceptional scalability, accommodating scenarios involving a large number of agents,

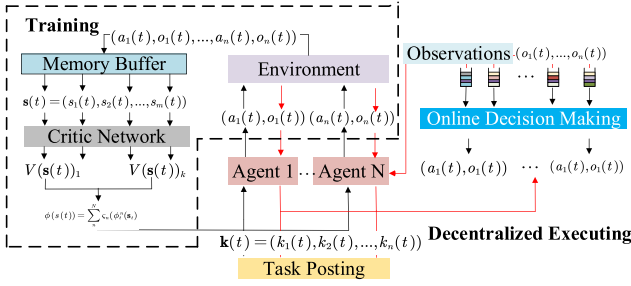


Fig. 2. Workflow of mixed MAPPO.

such as those found in crowd-edge environments. Importantly, it maintains consistent performance across diverse agents, ensuring robust and reliable results. Implementing MAPPO is straightforward and user-friendly. While certain RL algorithms with complex models (e.g., TD3) can be applied to the crowd-edge task offloading problem, their intricate design logic and stringent input data requirements often result in reduced efficiency when addressing this specific challenge. MAPPO offers a more streamlined and accessible solution.

Critic training: The critic network fixes the policy $\pi(\mathbf{a}|\mathbf{o})$, $\mathbf{a} \in \mathcal{A}$, $\mathbf{o} \in \mathcal{O}$ and approximates the value function $\beta_t(s_t)$ by minimizing the bellman error. Let θ denote the weights of the critic network, the $\beta_t(s_t)$ can be denoted by a nonlinear function $\phi_\theta(\cdot)$. The purpose of the critic network is to learn a $\phi_\theta(s_t)$ approximates the real $\beta_t(s_t)$. Let $\text{Pr}^\pi(s_{t+1}|s_t) := \sum_{\mathbf{a}_t \in \mathcal{A}} \pi(\mathbf{a}_t|\mathbf{o}(t)) \text{Pr}(\mathbf{o}(t)|\mathbf{a}_t, s_t) \text{Pr}\{s_{t+1}|\mathbf{a}_t, \mathbf{o}(t)\}$ be the probability of state s_t transit to state s_{t+1} given the policy $\pi(\mathbf{a}|\mathbf{o})$, where $\mathbf{a}_t = (\mathbf{w}(t), \mathbf{x}(t))$. Then, the objective of learning $\phi_\theta(s_t)$ is to minimize the mean square projected bellman error,

$$\begin{aligned} \min_{\theta} f'(\theta) \\ = \|B_\phi(\phi_\theta(s_t) - \text{Pr}^\pi(s_{t+1}|s_t)\phi_\theta(s_t) - r(\mathbf{w}(t), \mathbf{x}(t)))\| \end{aligned} \quad (25)$$

where B_ϕ is the project operator. Accordingly, the weight update in the critic network can be given by

$$\theta(t+1) = \theta(t) - \nabla_{\theta} f'(\theta)$$

In the multi-agent scenario, each actor outputs the policy based on their local observations. When we use a single critic network in the learning framework, we observe highly dynamic value outputs due to the frequent fluctuation failure to have global knowledge of the status variation. Such a dynamic on estimated value impairs learning performance and slows the convergence. To tackle the approximation error caused by these multi-agent features, we apply multiple critic networks with different values of initial states and learning rate and use the weight averaged sum of these outputs to reduce the bias. Specifically, each critic network independently estimates the value $V(s(t))$ using different initial parameters. In each episode, the estimated value $V(s(t))$ is derived by a linear combinations of multiple $V(s(t))_n$:

$$\phi(s(t)) = \sum_n^N \varsigma_n \phi_t^n(s_t)$$

where ς_n is a the weight and $\sum_n^N \varsigma_n = 1$. The utilization of the averaged sum of multiple critic networks is aimed at enhancing training performance. Each critic network individually estimates the state-action value. By averaging the outcomes from these individual critics, we can potentially mitigate the influence of potential variants in the multi-agent scenarios.

Actor training: Mixed MAPPO distributedly trains actor model for each agent. Each task's policy at t is parameterized as $\pi_{\omega_i(t)}(\mathbf{a}|\mathbf{o}_i) : \mathcal{O} \rightarrow \mathcal{A}$, where the $\omega_i(t)$ is the network parameters for each i .

To train the actor-network, we first define the clip function as follows:

$$\begin{aligned} C \left(\frac{\pi_{\omega_i(t)}(\mathbf{a}_t|\mathbf{o}_i)}{\pi_{\omega_i(t-1)}(\mathbf{a}_t|\mathbf{o}_i)} \right) \\ = \text{clip} \left(\frac{\pi_{\omega_i(t)}(\mathbf{a}_t|\mathbf{o}_i)}{\pi_{\omega_i(t-1)}(\mathbf{a}_t|\mathbf{o}_i)}, 1 - \epsilon_3, 1 + \epsilon_3 \right) \end{aligned} \quad (26)$$

where ϵ_3 is a hyper-parameter setting to 0.1. This function limits the value of $\frac{\pi_{\omega_i(t)}(\mathbf{a}_t|\mathbf{o}_i)}{\pi_{\omega_i(t-1)}(\mathbf{a}_t|\mathbf{o}_i)}$ within the interval $[1 - \epsilon_3, 1 + \epsilon_3]$. We further define the objective function for actor network updating:

$$\begin{aligned} \mathcal{L}^{Clip}(\omega_i(t)) \\ = \bar{E}_t \left[\min \left(\pi_{\omega_i(t-1)}^{\omega_i(t)} I(t, \mathbf{a}_t), C \left(\pi_{\omega_i(t-1)}^{\omega_i(t)} \right) I(t, \mathbf{a}_t) \right) \right] \end{aligned} \quad (27)$$

where $\pi_{\omega_i(t-1)}^{\omega_i(t)} := \frac{\pi_{\omega_i(t)}(\mathbf{a}_t|\mathbf{o}_i)}{\pi_{\omega_i(t-1)}(\mathbf{a}_t|\mathbf{o}_i)}$, \bar{E}_t denotes the average over time slots, The actor network's weights are updated by the following iterations, $I(t, \mathbf{a}_t)$ is the advantage function of the sampled trajectory $(\mathbf{a}_t, s_t, \dots, \mathbf{a}_T, s_T)$ denoted by

$$I(t, \mathbf{a}_t) := \sum_t^T \phi^{t-1}(\beta_t(s_t, \mathbf{a}_t) - \beta_t(s_t))$$

Accordingly, the actor network updates the parameters $\omega_i(t)$ by maximizing the objective function:

$$\omega_i(t+1) = \arg \max_{\omega} \mathcal{L}^{Clip}(\omega_i(t)) \quad (28)$$

After reaching the iteration criterion, the model will be forwarded to each device. The pseudo-code of training mixed MAPPO is shown in *Algorithm 2*. Due to the time-varying feature of the edge-crowd computing environment, we assume that the agent's decisions are temporally correlated, hence requiring a data buffer to store these states and update them during processing. Both the critic and actor-network includes recurrent neural networks (RNNs). This design requires a data buffer to store the time-series trajectory to train the RNNs.

Task offloading by mixed MAPPO: Our task offloading system efficiently offloads tasks to the crowd-edge by employing our proposed mixed MAPPO approach. Each MD is treated as an agent and is equipped with a well-trained actor network responsible for making informed offloading decisions. Specifically, when an MD generates an offloading request, it first assesses the current state denoted as $\mathbf{o}_i(t)$. Subsequently, the MD utilizes the policy $\pi^i_{\omega_i^*(t)}(\mathbf{a}_t|\mathbf{o}_i(t))$ to determine the action, represented as $\mathbf{w}'_i(t+1)$. Once the task is processed, the system caches and

Algorithm 2: Mixed MAPPO Training.

Input: $\theta(t) := \{\theta_i(t)\}_{i \in N}$, $\omega(t) := \{\omega_i(t)\}_{i \in N}$,
 $t = 0, \epsilon, s_0$;
Output: $\{\theta_i^*\}_{i \in N}, \{\omega_i^*\}_{i \in N}$

- 1 Initiate a memory buffer D;
- 2 Initiate a initial police $\pi_{\omega_i(t)}(\mathbf{a}_t | \mathbf{o}_i)$;
- 3 **while** $\|\theta(k) - \theta(k-1)\|^2 \geq \epsilon$ **do**
- 4 $t=0$;
- 5 **while** $t \leq T$ **do**
- 6 **foreach agent** $i \in \{1, \dots, N\}$ **do**
- 7 Derive $\{\mathbf{o}_i(t), \mathbf{w}_i(t), x_i(t), r(\mathbf{w}_i(t), x_i(t))\}$
 by the policy $\pi_{\omega_i(t)}(\mathbf{a} | \mathbf{o}_i)$;
- 8 **end**
- 9 **end**
- 10 **foreach agent** $i \in \{1, \dots, N\}$ **do**
- 11 Get a trajcetory
 $\{\mathbf{o}_i(t), \mathbf{w}_i(t), x_i(t), r(\mathbf{w}_i(t), x_i(t))\}_{t=1}^T$
- 12 **end**
- 13 **while** $t \leq T$ **do**
- 14 **foreach critic network do**
- 15 Compute $\phi_t^n(s_t)$ by minimiziing (24);
- 16 **end**
- 17 Compute the average sum of $\phi_t^n(s_t)$ to derive
 $\phi_t(s_t)$;
- 18 **end**
- 19 Derive the $\{\phi_t(s_t)\}_{t=1}^T$;
- 20 Compute the advantages $\{I(t, \mathbf{a}_t)\}_{t=1}^T$;
- 21 Store the trajectory
 $\{\{\mathbf{o}_i(t), \mathbf{w}_i(t), x_i(t), r(\mathbf{w}_i(t), x_i(t))\}^N_{i=1}\}_{t=1}^T$,
 $\{\phi_t(s_t), I(t, \mathbf{a}_t)\}_{t=1}^T$ into memory buffer D;
- 22 Sample a batch B of trajectories from D;
- 23 **foreach agent** $i \in \{1, \dots, N\}$ **do**
- 24 $\omega_i(k+1) =$
 $\frac{1}{B} \sum_B (\arg \max_{\omega} \mathcal{L}_b^{Clip}(\omega_i(k)))_{b \in B}$;
- 25 Update RNN hidden states;
- 26 **end**
- 27 **foreach critic network do**
- 28 Update the parameter
 $\theta(k+1) = \theta(k) - \nabla_{\theta} f'(\theta)$
- 29 Update RNN hidden states;
- 30 **end**
- 31 $\omega^* \leftarrow \omega(t)$;
- 32 **final** ;

consolidates the delay information. This data is then collected and used for future learning within the centralized training framework.

VI. PERFORMANCE EVALUATION

In this section, we first introduce the simulation settings and the numerical results by comparing mixed MAPPO with several state-of-the-art MARL algorithms. Besides, we also evaluate the performance of mixed MAPPO in a prototype system.

A. Simulation Settings

We consider a wireless communication scenario with $300 \times 900 \text{ m}^2$, which consists of three edge computing servers and six MD clusters. Each MD cluster consists of 5-10 MDs [39]. Each edge server connects a base station (BS), and BS is connected via wired links. We set the wired and wireless links' capacity to 30 Mbps and 10 Mbps, respectively. The simulation time is set to 3000s and each time slot equals 1s.

At each time slot, we generate tasks with random resource requirements and data size. The task generation rate follows a Poisson distribution λ_1 [40]. The time between two consecutive arriving tasks follows a uniform distribution over the range $U[5, 20]$. For each task, the computation resource requirement $s_i(t)$ ³ and delivering data $o_i(t)$ (The delivered data is in a unit of data size, i.e., 1MB) follow Poisson distributions λ_2 and λ_3 , respectively.

We set three different cases of task generation to verify mixed MAPPO's performance when processing diversified computational applications: (1) Case I, for each MD, λ_1, λ_2 , and λ_3 are set to 1, 24 and 6, respectively. Tasks in such case with large computation resource demand but high arriving interval, i.e., large scale deep learning tasks; (2) Case II represents media data processing scenario, i.e., live video transcoding and image processing. In this case, λ_1, λ_2 , and λ_3 are set to 6, 6, and 1, respectively; (3) In Case III, λ_1, λ_2 and λ_3 are set to 12, 3 and 0.5, respectively. In this case, tasks have very small computational resource demand but with the highest arrival rate frequency, i.e., crowdsensing.

B. Numerical Results

We have executed a comprehensive set of simulation tests utilizing the previously mentioned simulation configurations. Our evaluation focuses on comparing the performance of our proposed mixed MAPPO (termed as M2PPO) with that of three distinct solutions: Multi-Agent Deep Q-learning (MADQN), vanilla MAPPO, and multi-agent A3C (MAA3C). These schemes were chosen as benchmarks due to their well-established reputation for robust performance, versatility, extensive adoption within the research community, open-source implementations, scalability, and comprehensive documentation. These attributes collectively make them ideal candidates for comparison with our proposed approach. Furthermore, the objective of evaluating the performance of vanilla MAPPO for ablation analysis is to assess the improvement in the averaged weighted sum on critic networks. The implementation details of each algorithm are given as follows:

- The Q-network applies in MADQN employs a 4-layer structure: (1) a linear layer with $\text{dim}(\text{input}) \times 32$; (2) a linear layer with dimension 32×96 ; (3) LSTM cell with 96×64 ; (4) a linear layer with $64 \times \text{dim}(\text{output})$.
- The critic and actor networks in MAA3C employ a 4-layer structure: (1) a linear layer with $\text{dim}(\text{input}) \times 32$; (2) a

³The requested computation resources is in a unit of computation resources, i.e., 1GHz CPU cycles with 1 second

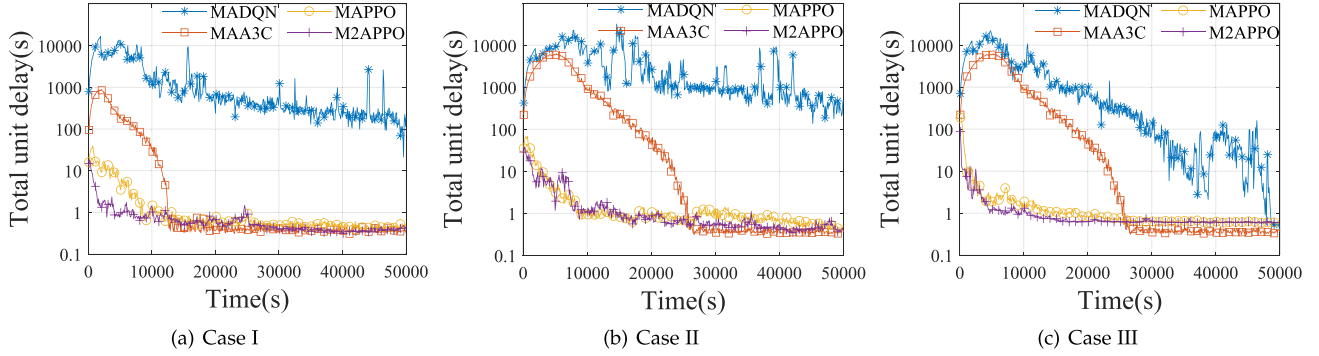


Fig. 3. Average execution delay(s) during the training: Case I, II, III.

linear layer with dimension 32×96 ; (3) an LSTM cell with 96×64 ; (4) a linear layer with $64 \times \text{dim}(\text{output})$.

- MAPPO and mixed MAPPO employ a 3-layer structure for actor-network: (1) a linear layer with $\text{dim}(\text{input}) \times 64$; (2) a linear layer with dimension 64×64 ; (3) A RNN with input dimension 64, hidden dimension 64; (4) a linear layer with $64 \times \text{dim}(\text{output})$. The number of critic network in mixed MAPPO is 5. The weight of the average is set to 0.2.

All these linear layers use the Rectified Linear Unit (ReLU) activation function. MAPPO consists of four critic networks and takes their weighted sum average as the output.

We evaluate the performance of mixed MAPPO with the above algorithms in terms of the unit delay when training and testing and load rate of computation nodes.

Unit Delay (UD) during training: Our training process involves datasets spanning 50,000 seconds. We define the unit delay as the processing time required for a unit of raw data in the task.

Fig. 3 illustrates the UD during training across three distinct scenarios. The figure reveals that all solutions are decreasing before eventually converging to stability. Notably, when compared to MADQN and MAA3C, our proposed mixed MAPPO demonstrates a faster convergence rate. For instance, in Fig. 3(a), mixed MAPPO achieves convergence at 8,000 seconds, while MAPPO converges at 10,000 seconds, and MAA3C at 12,000 seconds, respectively. MADQN, on the other hand, maintains a significantly higher level of delay and performs notably worse than the other two solutions in cases II and III, failing to reach the optimum during the simulations. This underperformance of MADQN can be attributed to its limited exploration capability, making it challenging to achieve good performance under diverse conditions. Additionally, it's worth noting that MADQN is typically suited for problems with restricted state and action spaces, such as chess or video games, whereas the task offloading problem is considerably more complex. MAA3C accelerates the training process through interactions between critic and actor networks. Furthermore, each MD employing MAA3C leverages multiple agents to explore various policies, a factor that contributes to its quicker convergence. Both mixed and vanilla MAPPO employ Actor-Critic (AC) networks to learn

optimal policies, exhibiting a similar convergence trend. The key distinction lies in our approach, where we derive the value function through a linear combination of results from multiple critic networks, thereby enhancing the robustness and accuracy of the value function.

Unit Delay (UD) during testing: To evaluate the unit delay, we conducted tests and presented the results in Fig. 4. In Case I, our proposed MAPPO demonstrates the best UD performance. This outcome can be attributed to the advantages of the MAPPO algorithm, as well as the enhancements achieved through averaged critic outputs. In cases II and III, several factors contribute to the similar performance levels observed among mixed MAPPO, MAPPO, and MAA3C: (1) The task arrival rate in cases II and III is relatively high compared to case I. When dealing with a high task arrival rate, the offloading policy needs frequent updates to search for the optimum, rendering convergence more challenging; (2) Partial observability also impacts task offloading performance due to limited knowledge of system dynamics. Our M2APPO addresses this by utilizing averaged critic outputs to estimate action performance, thereby improving the accuracy of value estimation. This higher accuracy provides better guidance for actor performance and consequently enhances offloading performance. The observed inferior performance of MADQN can be attributed to inadequate model training, making it challenging to offload tasks effectively to computation units with higher rewards.

Loss Convergence (LC): Fig. 5 showcases the loss variant of critic under three different cases: (1) Using single critic, methods such as MAPPO apply this policy to train the critic network; (2) using the minimum output, methods such as TD3 apply this policy to train the critic network; (3) using the average output, our proposed method use the average output of the multiple critic networks. From the figure, it is evident that in the context of task offloading, using the average output yields a slight improvement over the single critic approach. On the other hand, the minimum output used by TD3 method, which is designed for continuous action spaces, exhibits fluctuations in the loss before eventually converging.

Loss value: We investigate the loss value during training the critic network. A small loss value is preferred since it indicates a higher accuracy on value function estimation.

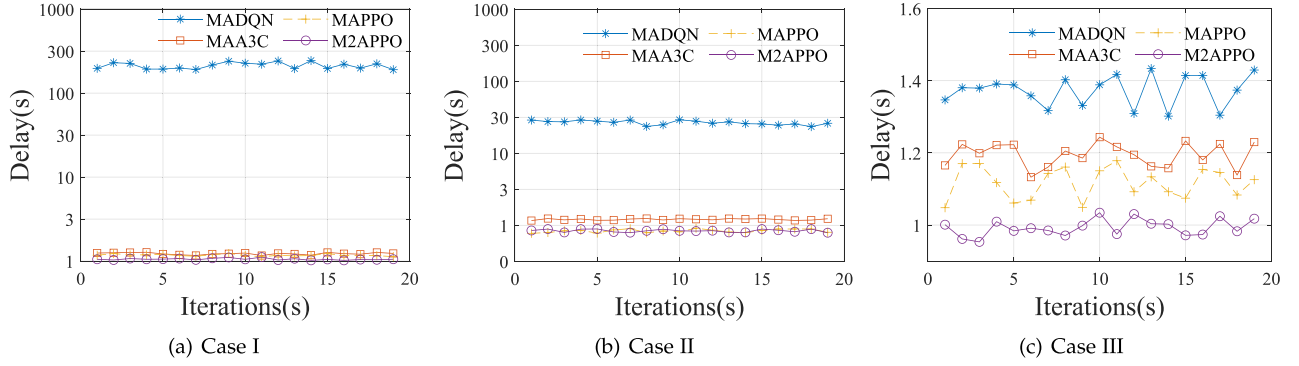


Fig. 4. Unit delay(s) during the test: Case I, II, III.

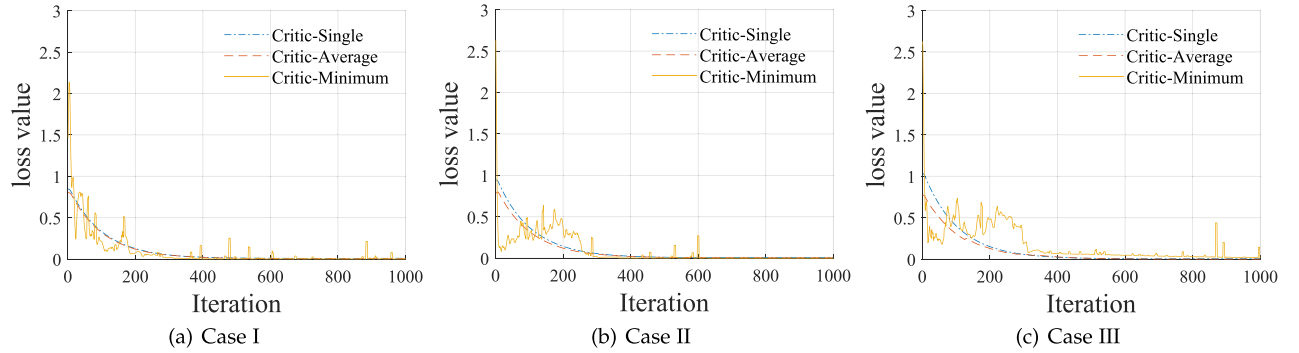


Fig. 5. Loss value during the training: Case I, II, III.

Load Rate (LR): We gauge the average load rate at each computation node across various scenarios. Our definition of load rate involves quantifying the ratio between pending computation resource requests and the computational capability of a given node. Excessive loads on Mobile Devices (MDs) can accelerate energy consumption, potentially discouraging end-users from contributing their resources. Conversely, an increased load on the edge can elevate processing overhead. Therefore, a well-balanced load is highly desirable. To ensure that all tasks can be adequately processed, we introduce a Cloud component in this scenario to handle requests that exceed the edge's capacity. From Fig. 6, we observe that mixed MAPPO consistently achieves superior load balancing among MDs across all three cases when compared to other solutions. In all approaches, case III exhibits the highest load rate on the device side. This is because tasks in case III require fewer computation resources than those in cases I and II, prompting the offloading policy to favor local processing by MDs.

In contrast to mixed MAPPO, MAPPO, and MAA3C, MADQN predominantly offloads tasks to MDs while underutilizing the edge. MAA3C exhibits a more aggressive approach, preferring edge processing. In M2APPO, each MD strives to maximize task processing efficiency while also considering the load on computation units and network link congestion. Consequently, M2APPO reduces overall processing delay while maintaining a balanced load distribution among different computation units.

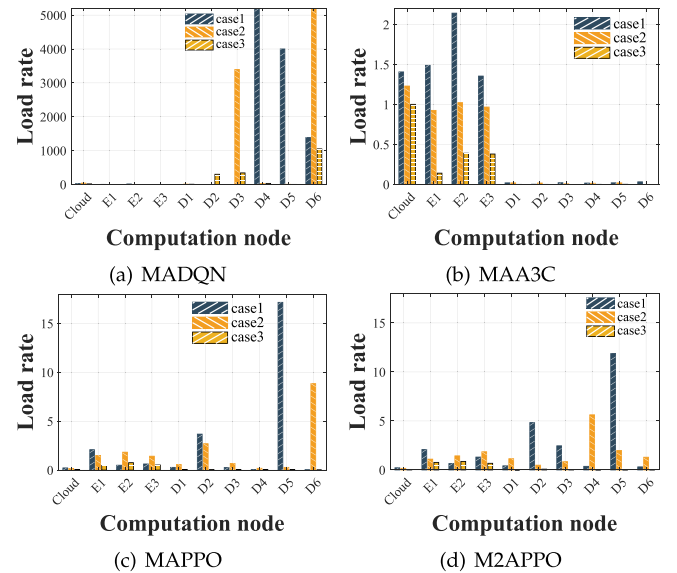


Fig. 6. Average load rate at different nodes.

Table III collectively presents the comprehensive performance evaluation, encompassing average unit delay (AD), average node load rate (ANLR), average edge-to-edge (E2E) link load rate, and average Edge-to-Device (E2D) link load rate across three distinct cases. Our definition of link load rate involves quantifying the ratio between pending delivery

TABLE III
PERFORMANCE OF FOUR APPROACHES IN CASES II AND III

Case I				
Approaches	AD	ANLR	E2E	E2D
MADQN	220	5190.76	0.06889	0.949
MAA3C	1.215	0.01887	0.11644	0.77633
MAPPO	1.18	3.3796	0.005	0.05833
M2APPO	1.08	3.56947	0.005	0.10286
Case II				
Approaches	AD	ANLR	E2E	E2D
MADQN	26.3	5839.45	58.26	0.32
MAA3C	1.185	0.013	0.14756	0.71667
MAPPO	0.84286	2.20613	0.01	0.07867
M2APPO	0.84286	1.89587	0.005	0.08867
Case III				
Approaches	AD	ANLR	E2E	E2D
MADQN	1.3673	289.27	0.13178	0.65267
MAA3C	1.18666	0.001	0.16533	0.60917
MAPPO	1.114173	0.3953	0.005	0.0515
M2APPO	1.01048	0.0307	0.005	0.08883

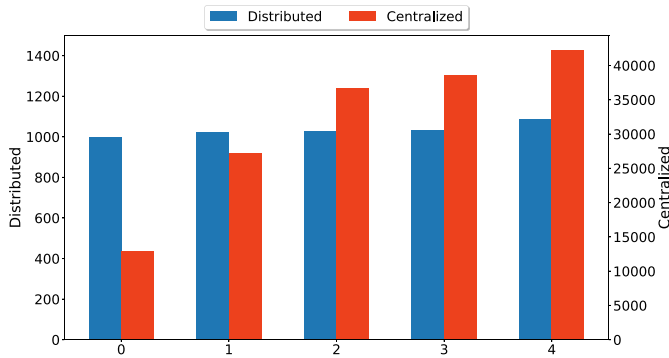


Fig. 7. Communication overhead.

data and available bandwidth. The results clearly illustrate that M2APPO consistently outperforms other solutions across all cases, achieving superior results in terms of E2E, E2D, and AD. This superior performance in AD aligns with the insights discussed in the context of Fig. 4, where M2APPO showcased lower delays. The lower E2E achieved by M2APPO can be attributed to its efficient utilization of MD resources, minimizing the need for frequent task scheduling among edge servers and thereby mitigating traffic pressures between them. Furthermore, M2APPO's consideration of network congestion while offloading tasks results in bandwidth savings, leading to lower values for both E2E and E2D.

Fig. 7 compares the communication overhead between our distributed offloading scheme and a centralized offloading scheme. In the centralized offloading approach, a central coordinator collects status information and distributes decisions to each agent. The communication overhead is quantified by the bandwidth consumed in transmitting this status information. As illustrated in the figure, the bars representing M2APPO exhibit a sub-linear increase in communication overhead as the number of nodes grows, demonstrating that our proposed method scales efficiently. In contrast, the centralized method shows a linear increase in communication overhead, indicating that as the number of nodes increases, the communication overhead for the centralized scheme escalates rapidly.

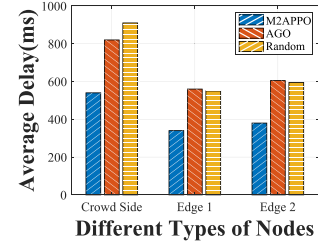


Fig. 8. Average execution delay.

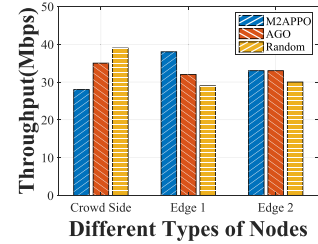


Fig. 9. Throughput for AGO.

C. Test on System Prototype

To further evaluate the performance of proposed tasks offloading based on mixed M2APPO in the real world, we build a prototype task offloading system based on our previously designed computation platform. This prototype system deploys across multiple physical machines. The prototype system leverages crowd-edge computing to enhance the transcoding of live streaming services. In this setup, each edge device and mobile device (MD) has the capability to transcode and render video content to the desired bitrate. The edge live streaming scenario serves as a quintessential use case for testing task offloading, as it demands both sufficient bandwidth and computational resources. This prototype system includes two edge servers. Devices and servers use FFmpeg to process video. Each node is installed with a resource monitor to record the CPU/GPU occupancy. The wired link is 100 Mbps. In MDs, a PyTorch-based learning framework is equipped to generate the offloading policy-based local and edge resource monitors. Each MD randomly generates video content to transcode. The length of the video content is a random variable $U[2, 10]$ s. Video has multiple representations: 1080p(60fps), 1080p(30fps), 720p(60fps), 720p(30fps), 480p(30fps).

Based on this prototype system, we compared the performance of mixed MAPPO against that of our previously proposed resource scheduling algorithm, AGO [38], as well as a random policy. The AGO algorithm makes offloading decisions by solving a stochastic network optimization problem and uses a random policy to determine node offloading transcoding requests.

Figs. 8 and 9 reveal the average delay and throughput at the different nodes, respectively. In Fig. 8, we observe that edge sides have the lowest delay since edge servers are near the clients. We also find that mixed MAPPO outperforms the other solutions on three different types of nodes. We measure the average throughput by the in and out data traffic per second.

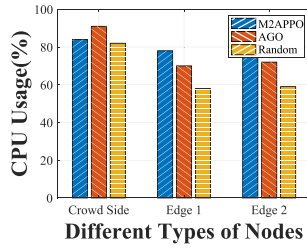


Fig. 10. CPU usage.

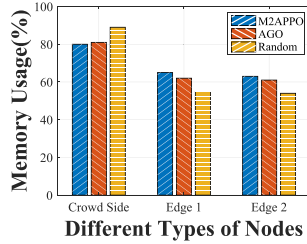


Fig. 11. Memory usage.

As shown in the figure, mixed MAPPO reduces the average throughput by almost 5% and 13% when compared with AGO and Random policy, respectively. This improvement is because our proposed solution can provide better-balanced task offloading which reduces the network traffic. Figs. 10 and 11 show the average CPU and memory usage. We observe that both the edge and client sides' resource usage of mixed MAPPO remains relatively high. Namely, mixed MAPPO offloads more tasks to the edge to reduce the processing delay. Since MDs' resource is cheap in usage and proximity to users, efficiently offloading tasks to MDs in mixed MAPPO reduces the processing delay and resource usage overhead. AGO achieves a higher load rate on the client side, which means more tasks have been offloaded to the client side. However, due to the inefficient task offloading, the average delay of AGO is still higher than that of the mixed MAPPO.

To evaluate the computation overhead on training the Mixed MAPPO, we use PyTorch and deploy the model on a desktop with two NVIDIA RTX 4090 GPUs and an Intel Xeon 6080P CPU. During training, the PyTorch process averagely occupied 62% GPUs and 15% CPU. When training the original MAPPO, the average occupancy of GPUs and CPU are 51% and 13%, respectively. We also test the inference latency using the trained model on mobile devices. We deploy the trained model on two smartphones Redmi K30 Ultra and iPhone 13. For these two devices, the time consumption of using the trained model to determine which computation unit to offload the tasks is 253.47 ms (Redmi K30 Ultra) and 126.15 ms (iPhone 13).

VII. CONCLUSION

In this paper, we introduce a novel approach for task offloading in crowd-edge systems. Our work begins by framing the crowd-edge task offloading challenge as a stochastic optimization problem, accounting for partial network observability. We then illustrate a method to estimate network status using

end-to-end delay, offering a reformulation of the optimization problem. To address the problem in practice, we present a multi-agent reinforcement learning algorithm called mixed MAPPO, which employs a decentralized approach to derive the optimal offloading policy.

We also conduct both numerical evaluations and simulation tests. Our numerical results reveal that mixed MAPPO outperforms state-of-the-art MARL algorithms when addressing the crowd-edge task offloading problem. Furthermore, our prototype-level simulations demonstrate that our algorithm surpasses existing task offloading solutions. Through our studies, we've observed that mobile devices within crowds exhibit cooperative and competitive behaviors, suggesting that game theory could provide valuable insights for further analyzing system dynamics.

In the future, we will consider using knowledge distillation and transfer learning methods to alleviate the computation overhead of using our algorithm. For example, we first train a teacher model on powerful computation units, such as servers. Knowledge distillation would then enable the transfer of learned knowledge to a student model with smaller parameter sizes, making it more suitable for resource-limited devices.

REFERENCES

- [1] M. Chen, B. Liang, and M. Dong, "Multi-user multi-task offloading and resource allocation in mobile cloud systems," *IEEE Trans. Wireless Commun.*, vol. 17, no. 10, pp. 6790–6805, Oct. 2018.
- [2] D. Liu, L. Khokhi, and A. Hafid, "Prediction-based mobile data offloading in mobile cloud computing," *IEEE Trans. Wireless Commun.*, vol. 17, no. 7, pp. 4660–4673, Jul. 2018.
- [3] H. Wang, H. Xu, H. Huang, M. Chen, and S. Chen, "Robust task offloading in dynamic edge computing," *IEEE Trans. Mobile Comput.*, vol. 22, no. 1, pp. 500–514, Jan. 2023.
- [4] H. A. Alameddine, S. Sharafeddine, S. Sebbah, S. Ayoubi, and C. Assi, "Dynamic task offloading and scheduling for low-latency IoT services in multi-access edge computing," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 3, pp. 668–682, Mar. 2019.
- [5] M. Langer, Z. He, W. Rahayu, and Y. Xue, "Distributed training of deep learning models: A taxonomic perspective," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 12, pp. 2802–2818, Dec. 2020.
- [6] A. S. Berahas, R. Bollapragada, N. S. Keskar, and E. Wei, "Balancing communication and computation in distributed optimization," *IEEE Trans. Autom. Control*, vol. 64, no. 8, pp. 3141–3155, Aug. 2019.
- [7] L. Zhao, Q. Wang, Q. Zou, Y. Zhang, and Y. Chen, "Privacy-preserving collaborative deep learning with unreliable participants," *IEEE Trans. Inf. Forensics Secur.*, vol. 15, no. 1, pp. 1486–1500, Sep. 2020.
- [8] Z. Zhang et al., "SecCL: Securing collaborative learning systems via trusted bulletin boards," *IEEE Commun. Mag.*, vol. 58, no. 1, pp. 47–53, Jan. 2020.
- [9] X. Wang, X. Chen, and W. Wu, "Towards truthful auction mechanisms for task assignment in mobile device clouds," in *Proc. IEEE Conf. Comput. Commun.*, 2017, pp. 1–9.
- [10] W. Zhang, N. Tang, D. Yang, R. Guo, H. Zhang, and X. Shen, "Det(Com)²: Deterministic communication and computation integration toward AIGC services," *IEEE Wireless Commun.*, vol. 31, no. 3, pp. 32–41, Jun. 2024.
- [11] M. Wang, C. Xu, X. Chen, L. Zhong, Z. Wu, and D. O. Wu, "BC-Mobile device cloud: A blockchain-based decentralized truthful framework for mobile device cloud," *IEEE Trans. Ind. Informat.*, vol. 17, no. 2, pp. 1208–1219, Feb. 2021.
- [12] B. Lin, Y. Huang, J. Zhang, J. Hu, X. Chen, and J. Li, "Cost-driven offloading for DNN-based applications over cloud, edge, and end devices," *IEEE Trans. Ind. Informat.*, vol. 16, no. 8, pp. 5456–5466, Aug. 2020.
- [13] M. Tang and V. W. S. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Trans. Mobile Comput.*, vol. 21, no. 6, pp. 1985–1997, Jun. 2022.

- [14] L. Ale, N. Zhang, X. Fang, X. Chen, S. Wu, and L. Li, "Delay-aware and energy-efficient computation offloading in mobile edge computing using deep reinforcement learning," *IEEE Trans. Cogn. Commun. Netw.*, vol. 7, no. 3, pp. 881–892, Sep. 2021.
- [15] W. Y. B. Lim et al., "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 2031–2063, Third Quarter, 2020.
- [16] T. Zhang and Q. Zhu, "Dynamic differential privacy for ADMM-based distributed classification learning," *IEEE Trans. Inf. Forensics Secur.*, vol. 12, no. 1, pp. 172–187, Jan. 2017.
- [17] A. Zavodovski, S. Bayhan, N. Mohan, P. Zhou, W. Wong, and J. Kangasharju, "DeCloud: Truthful decentralized double auction for edge clouds," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst.*, 2019, pp. 2157–2167.
- [18] R. S. Sutton, "Policy gradient method for reinforcement learning with function approximation," in *Proc. 1998 IEEE Int. Conf. Robot. Automat.*, 2000, pp. 1057–1063.
- [19] M. Volodymyr et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 7540, no. 518, pp. 529–533, 2019.
- [20] Y. Zhang et al., "Asynchronous methods for model-based reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1338–1347.
- [21] S. Iqbal and F. Sha, "Actor-attention-critic for multi-agent reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 2961–2970.
- [22] J. Foerster et al., "Counterfactual multi-agent policy gradients," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 2974–2982.
- [23] Z. Hong, W. Chen, H. Huang, S. Guo, and Z. Zheng, "Multi-hop cooperative computation offloading for industrial IoT-Edge-Cloud computing environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 12, pp. 2759–2774, Dec. 2019.
- [24] Y. He, J. Ren, G. Yu, and Y. Cai, "D2D communications meet mobile edge computing for enhanced computation capacity in cellular networks," *IEEE Trans. Wireless Commun.*, vol. 18, no. 3, pp. 1750–1763, Mar. 2019.
- [25] J. Wen, C. Ren, and A. K. Sangaiah, "Energy-efficient device-to-device edge computing network: An approach offloading both traffic and computation," *IEEE Commun. Mag.*, vol. 56, no. 9, pp. 96–102, Sep. 2018.
- [26] H. Xing, L. Liu, J. Xu, and A. Nallanathan, "Joint task assignment and resource allocation for D2D-enabled mobile-edge computing," *IEEE Trans. Commun.*, vol. 67, no. 6, pp. 4193–4207, Jun. 2019.
- [27] D. Yi, X. Zhou, Y. Wen, and R. Tan, "Efficient compute-intensive job allocation in data centers via deep reinforcement learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 6, pp. 1474–1485, Jun. 2020.
- [28] M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu, and W. Zhuang, "Learning-based computation offloading for IoT devices with energy harvesting," *IEEE Trans. Veh. Technol.*, vol. 68, no. 2, pp. 1930–1941, Feb. 2019.
- [29] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4005–4018, Jun. 2019.
- [30] Y. Zhang, B. Di, Z. Zheng, J. Lin, and L. Song, "Distributed multi-cloud multi-access edge computing by multi-agent reinforcement learning," *IEEE Trans. Wireless Commun.*, vol. 20, no. 4, pp. 2565–2578, Apr. 2021.
- [31] C. Liu, F. Tang, Y. Hu, K. Li, Z. Tang, and K. Li, "Distributed task migration optimization in MEC by extending multi-agent deep reinforcement learning approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 7, pp. 1603–1614, Jul. 2021.
- [32] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, and H. V. Poor, "Cooperative task offloading and block mining in blockchain-based edge computing with multi-agent deep reinforcement learning," *IEEE Trans. Mobile Comput.*, vol. 22, no. 4, pp. 2021–2037, Apr. 2023.
- [33] L. Wang, K. Wang, C. Pan, W. Xu, N. Aslam, and L. Hanzo, "Multi-agent deep reinforcement learning-based trajectory planning for multi-UAV assisted mobile edge computing," *IEEE Trans. Cogn. Commun. Netw.*, vol. 7, no. 1, pp. 73–84, Mar. 2021.
- [34] M. Wang, C. Xu, X. Chen, H. Hao, L. Zhong, and D. O. Wu, "Design of multipath transmission control for information-centric Internet of Things: A distributed stochastic optimization framework," *IEEE Internet Things J.*, vol. 6, no. 6, pp. 9475–9488, Dec. 2019.
- [35] S. Yao et al., "Blockchain-empowered collaborative task offloading for cloud-edge-device computing," *IEEE J. Sel. Areas Commun.*, vol. 40, no. 12, pp. 3485–3500, Dec. 2022.
- [36] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv: 1707.06347*.
- [37] M. Neely, *Stochastic Network Optimization With Application to Communication and Queueing Systems*. San Rafael, CA, USA: Morgan & Claypool, 2010.
- [38] X. Chen, C. Xu, M. Wang, Z. Wu, L. Zhong, and L. A. Grieco, "Augmented queue-based transmission and transcoding optimization for live-cast services based on CER integration," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 31, no. 11, pp. 4470–4484, Nov. 2021.
- [39] M. Gao, R. Shen, L. Shi, W. Qi, J. Li, and Y. Li, "Task partitioning and offloading in DNN-task enabled mobile edge computing networks," *IEEE Trans. Mobile Comput.*, vol. 22, no. 4, pp. 2435–2445, Apr. 2023.
- [40] X. Ma, A. Zhou, S. Zhang, Q. Li, A. Liu, and S. Wang, "Dynamic task scheduling in cloud-assisted mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 22, no. 4, pp. 2116–2130, Apr. 2023.



Su Yao received the PhD degree from the National Engineering Laboratory for Next Generation Internet, Beijing Jiaotong University. Currently, he serves the Beijing National Research Center for Information Science and Technology (BNRist), Tsinghua University. His research interests include future network architecture, IoT security.



Mu Wang received the PhD degree in computer technology from the Beijing University of Posts and Telecommunications (BUPT), in 2020. He is currently serves as an associate researcher with the State Key Laboratory of Network and Switching Technology. His research interests include wireless communications and multimedia communications.



Ju Ren (Senior Member, IEEE) received the PhD degrees all in computer science from Central South University, China, in 2016. Currently, he is an associate professor with the Department of Computer Science and Technology, Tsinghua University, China. His research interests include Internet-of-Things, edge computing, security, and privacy. He currently serves as an associate editor of *IEEE Transactions on Cloud Computing*, etc. He received IEEE TCSC Early Career Researcher Award (2019), and the IEEE ComSoc Asia-Pacific Best Young Researcher Award (2021).



Tianyu Xia received the bachelor's degree from Tianjin University, and the master's degree from Peking University. Currently, he is an AI algorithm engineer with Ant Group. His research interests are mainly neural networks on mobile devices, privacy-preserving machine learning, and multimodal models for UI interfaces.



Weiqiang Wang received the PhD degree in materials engineering from the University of Southern California. He is the director of the Machine Intelligent Department, Security and Risk Management Group, Ant Group. His current main focus is to develop high performance machine learning models in the new generation risk control engine.



Mingwei Xu (Senior Member, IEEE) received the BSc and PhD degrees from Tsinghua University. He is a full professor with the Department of Computer Science, Tsinghua University. His research interests include computer network architecture, high-speed router architecture, and network security.



Ke Xu (Fellow IEEE) received the PhD degree from the Department of Computer Science & Technology, Tsinghua University, where he serves as a full professor. He has published more than 100 technical papers and holds 20 patents in the research areas of next generation Internet and P2P systems.



Hongke Zhang (Fellow, IEEE) received the PhD degree in communication and information systems from the University of Electronic Science and Technology of China, Chengdu, China, in 1992. He is currently a professor with the School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing, China, where he currently directs the National Engineering Center of China on Mobile Specialized Network. He is an academician of China Engineering Academy, Beijing, and the co-director of the PCL Research Center of Networks and Communications, Peng Cheng Laboratory, Shenzhen, China. His current research interests include architecture and protocol design for the future Internet and specialized networks. He currently serves as an associate editor of *IEEE Transactions on Network And Service Management* and *IEEE Internet of Things Journal*.