Multi-Agent Deep Reinforcement Learning for Energy-Efficient Computation Offloading in Mobile Edge Computing

***Abstract*—**

***Index Terms*—Mobile edge computing, Resource allocation, Computation offloading, Des-POMDP, Multi-agent deep reinforcement learning**

I. Introduction

MOBILE edge computing (MEC) often suffers from the dynamic and unknown nature of the environment such as time-varying conditions, heterogeneous devices, and frequent communication requests, imposing significant challenges on improving system performance. To meet the rapidly growing demands of computation-intensive and time-sensitive applications, reinforcement learning (RL) [1] has been proposed as an effective tool to establish low-latency and energy-efficient networks. RL enables network entities to interact with the environment and learn an optimal decision-making policy, usually modeled as a markov decision process (MDP) [2].

Several research studies have explored RL-based approaches to addres key issues in MEC. To optimally adapts wireless resource allocations, Huang *et al.* in [3] proposed an online offloading framework capable of attaining near-optimal decisions. Zhao *et al.* in [4] developed a computation offloading algorithm to address competition for wireless channels. To handle deadline-constrained computational tasks, Tang *et al.* in [5] introduced a distributed offloading algorithm that manages uncertain workload dynamics at the edge nodes. Zhou *et al.* in [6] investigated the joint optimization of computation offloading and resource allocation to minimize energy consumption across the entire MEC. Sun *et al.* in [7] tackled both computation offloading and service caching problems, proposing a hierarchical DRL framework to minimize long-term average service delay. We also in [8] studied the computation offloading problem under strict task processing deadlines and energy constraints, proposing a distributed algorithm to maximize each user's long-term quality of experience (QoE) individually.

However, MEC networks typically involve multiple network entities interacting with each other, making single-agent RL inefficient and sometimes inapplicable. Single-agent RL learns its decision-making policy independently and treats other agents as part of the environment, which may cause the non-stationarity issue [9] and significantly reduce learning efficiency and network performance. In real-world network environments, each network entity must make action decisions based on its local observation, which only provides partial observation of the global network state. Thus, the independent learning network entity will struggle to incorporate the policies of other entities and mitigate the non-stationarity issue.

To tackle these challenges, researchers have turned to the use of multi-agent RL (MARL) methods in MEC, where multiple agents collaborate to make distributed decisions through global optimization. Yang *et al.* in [10] investigated the cooperative task offloading problem and proposed a multi-agent actor-critic framework to reduce the task computation delay. Nguyen *et al.* in [11] introduced a multi-agent deep deterministic policy gradient algorithm to maximize system utility by jointly optimizing offloading decisions, channel selection, transmit power allocation, and computational resource allocation in blockchain-based MEC. To optimize the overall offloading strategy in heterogeneous MEC, Gao *et al.* in [12] proposed a MARL-based offloading algorithm, where multiple agents work together to address the resource competition problem. Tan *et al.* in [13] formulated the problem of maximizing the number of offloaded tasks and ensuring offloading fairness, conceiving a MARL framework that leverages communication among agents. To minimize total energy consumption, Munir *et al.* [14] developed a semi-distributed approach using a multi-agent meta-RL framework for self-powered MEC.

II. Motivation and Contributions

Devise-edge task offloading. Efficient task offloading is crucial to ensure seamless resource distribution in MEC. Device-edge task offloading enables devices to independently make decisions on offloading resource-intensive tasks to nearby edge servers, fostering efficient utilization of available resources.

Edge-edge task offloading. Task offloading leverages edge-edge collaborations, where tasks initially received by a local edge server can be offloaded to neighboring servers with underutilized resources, ensuring better resource utilization. The task offloading decision-making process focuses on efficiently distributing tasks among edge servers. Offloading tasks between edge servers requires communication resources and may introduce additional transmission delay, which should be taken into account when designing offloading strategies.

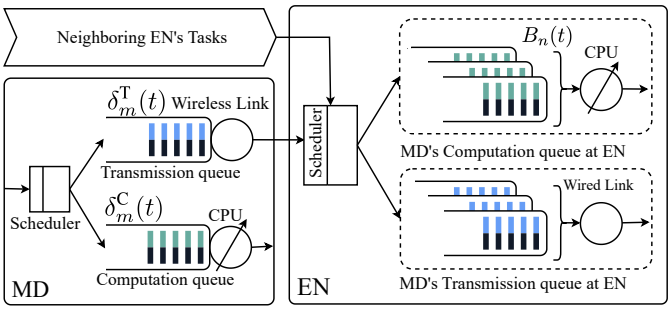

Fig. 1. An illustration of MD $i \in \mathcal{I}$ and EN $j \in \mathcal{J}$ in the MEC system.

III. System Model and Problem Formulation

We investigated a MEC network based on queueing system, including a set of MDs $\mathcal{I} = \{1, 2, \cdot, i, \cdot, I\}$, and a set of ENs $\mathcal{J} = \{1, 2, \cdot, j, \cdot, J\}$ in coverage areas at time slot t, where time is regarded as a specific episode containing a series of time slots $\mathcal{T} = \{1, 2, \cdot, t, \cdot, T\}$, each representing a duration of τ seconds. As shown in Fig. 1, multiple first-in-first-out (FIFO) queues are applied as computation and communication models for MDs and ENs. Each MD has two separate queues to manage tasks for local processing or dispatching to ENs. Similarly, each EN $j \in \mathcal{M}$ consists of two associated queues for each MD, designed for processing tasks or dispatching them to neighboring ENs. The computing task $k_i(t)$ is generated by MD $i \in \mathcal{I}$ at time slot $t \in \mathcal{T}$ is described as $\{\lambda_i^{\mathrm{S}}(t), \lambda_i^{\mathrm{C}}(t), \lambda_i^{\mathrm{D}}(t)\}$, where $\lambda_i^{\mathrm{S}}(t)$, $\lambda_i^{\mathrm{C}}(t)$, and $\lambda_i^{\mathrm{D}}(t)$ denote the task $k_i(t)$'s data size, the number of CPU cycles required to execute, and the maximum delay tolerated by MD i's task, respectively. We define the binary variable $x_i(t)$ to represent the offloading decisions for MD $i \in \mathcal{N}$, where $x_i(t) = 0$ indicates that the task is assigned to the computation queue, and $x_i(t) = 1$ denotes assignment to the transmission queue. We also defined vector $\boldsymbol{y}_j(t) = (y_{i,j}(t))_{i\in\mathcal{N}}$ to denote the EN i's offloading decision at time t, where $y_{i,j}(t)$ indicates whether arrived task from MD i to EN j is assigned to the associated computation queue ($y_{i,j}(t) = 0$) or associated communication queue for dispatching to neighboring ENs ($y_{i,j}(t) = 1$).

MD's computation model: We model local execution consisting of a computation queue and MD processor. Let f_i be MD i's processing power. When task $k_i(t)$ enters the computation queue at time t, $l_i^{\mathrm{C}}(t)$ represents when the task is processed or dropped. If the queue is empty, $l_i^{\mathrm{C}}(t) = 0$. Let $\delta_i^{\mathrm{C}}(t)$ denote the number of remaining time slots before processing task in the computation queue:

$$\delta_i^{\mathrm{C}}(t) = \left[\max_{t' \in \{0,1,\ldots,t-1\}} l_i^{\mathrm{C}}(t') - t + 1 \right]^+. \tag{1}$$

$\delta_i^{\mathrm{C}}(t)$ is the waiting time for task before processing. If task is assigned to the computation queue for local processing at time t, it's processed by time $l_i^{\mathrm{C}}(t)$:

$$l_i^{\mathrm{C}}(t) = \min\left\{ t + \delta_i^{\mathrm{C}}(t) + \left\lceil \frac{\lambda_i^s(t)}{f_i\tau/\lambda_i^c(t)} \right\rceil - 1, t + \lambda_i^d(t) - 1 \right\}. \tag{2}$$

MD's transmission model: We assume tasks in the transmission queue are sent to appropriate ENs through the MD wireless interface. The transmission rate between MD i and EN j at time t is $r_{i,j}(t)$. If task $k_i(t)$ is offloaded in time slot t, $l_i^{\mathrm{T}}(t)$ represents when the task is dispatched or dropped. $\delta_i^{\mathrm{T}}(t)$ denotes waiting time slots before transmission. It should be noted that MD i computes the value of $\delta_i^{\mathrm{T}}(t)$ before making a decision. The value of $\delta_i^{\mathrm{T}}(t)$ is computed as follows:

$$\delta_i^{\mathrm{T}}(t) = \left[\max_{t' \in \{0,1,\ldots,t-1\}} l_i^{\mathrm{T}}(t') - t + 1 \right]^+, \tag{3}$$

$\delta_i^{\mathrm{T}}(t)$ depends on $l_i^{\mathrm{T}}(t)$ for $t' < t$. If MD i schedules task at time t, it's dispatched or dropped at $l_i^{\mathrm{T}}(t)$:

$$l_i^{\mathrm{T}}(t) = \min\left\{ t + \delta_i^{\mathrm{T}}(t) + \left\lceil \frac{\lambda_i^s(t)}{r_{i,j}(t)\tau} \right\rceil - 1, t + \lambda_i^d(t) - 1 \right\}. \tag{4}$$

EN's computation model Tasks which are assigned to the associate MD's computation queue at EN execute by the allocated EN's computional resource. Let $\eta_{i,j}^{\mathrm{C}}(t)$ represent the computation queue length of MD i at the end of time slot t in EN j. The set of active queues for computation at EN j in time slot t is given by $\mathcal{B}_j^{\mathrm{C}}(t) = \{i \mid i \in \mathcal{N}, \lambda_i^{\mathrm{S}}(t) > 0 \text{ or } \eta_{i,j}^{\mathrm{C}}(t-1) > 0\}$, where $b_j^{\mathrm{C}}(t) \triangleq |\mathcal{B}_j^{\mathrm{C}}(t)|$. We divide EN's processing power among active queues using a generalized processor sharing method [15]. Let f_j^{C} (in cycles per second) denote EN j's computational capacity. EN j allocates $f_j^{\mathrm{C}}/(\lambda_i^{\mathrm{C}}(t) b_j^{\mathrm{C}}(t))$ computational capacity to each MD $i \in \mathcal{B}_j(t)$ in time slot t. To calculate the computation queue length for MD i at EN j, we define $\omega_{i,j}(t)$ (in bits) as the number of bits from dropped tasks at the end of time slot t. The backlog of the queue, referred to as $\eta_{i,j}^{\mathrm{C}}(t)$ is given by:

$$\eta_{i,j}^{\mathrm{C}}(t) = \left[\eta_{i,j}^{\mathrm{C}}(t-1) + \lambda_i^{\mathrm{S}}(t) - \frac{f_j^{\mathrm{C}}\tau}{\lambda_i^{\mathrm{C}}(t) b_j^{\mathrm{C}}(t)} - \omega_{i,j}^{\mathrm{C}}(t) \right]^+. \tag{5}$$

We also define $l_{i,j}^{\mathrm{E}}(t) \in \mathcal{T}$ as the time slot during which the offloaded task $k_{i,j}(t)$ is either processed or dropped by EN j. Given the uncertain workload ahead at EN j, neither MD i nor EN j has information about $l_{i,j}^{\mathrm{E}}(t)$ until the corresponding task is either processed or dropped. Let $\hat{l}_{i,j}^{\mathrm{E}}(t)$ represent the time slot at which the execution of task $k_{i,j}(t)$ starts, we have:

$$\hat{l}_{i,j}^{\mathrm{E}}(t) = \max\{t, \max_{t' \in \{0,1,\ldots,t-1\}} l_{i,j}^{\mathrm{E}}(t') + 1\}, \tag{6}$$

where $\hat{l}_{i,j}^{\mathrm{E}}(0) = 0$. Indeed, the initial processing time slot of task $k_{i,j}(t)$ at EN should not precede the time slot when the task was enqueued or when the previously arrived tasks were processed or dropped. Therefore, $l_{i,j}^{\mathrm{E}}(t)$ is the time slot that satisfies the following constraints.

$$\sum_{t'=\hat{l}_{i,j}^{E}(t)}^{l_{i,j}^{E}(t)} \frac{f_j^E}{\lambda_{i,j}^{\mathrm{C}}(t) b_j^{\mathrm{C}}(t')} \geq \lambda_{i,j}^{\mathrm{S}}(t) > \sum_{t'=\hat{l}_{i,j}^{E}(t)}^{l_{i,j}^{E}(t)-1} \frac{f_j^E}{\lambda_{i,j}^{\mathrm{C}}(t) b_j^{\mathrm{C}}(t')}. \tag{7}$$

In particular, the total processing capacity that EN j allocates to MD i from the time slot $\hat{l}_{i,j}^{\mathrm{E}}(t)$ to the time slot $l_{i,j}^{\mathrm{E}}(t)$ should exceed the size of task $\lambda_{i,j}^{\mathrm{S}}(t)$. Conversely, the total allocated processing capacity from the time slot $\hat{l}_{i,j}^{\mathrm{E}}(t)$ to the time slot $l_{i,j}^{\mathrm{E}}(t) - 1$ should be less than the task's size. We assume that each EN can perceive the number of active queues $b_j^{\mathrm{C}}(t)$ in the previous time time slot by broadcasting messages at

each ending time by the ENs. The EN calculates the bits processed by active queues and estimates the bits discarded due to the maximum tolerated delay. Since the number of active computation queues varies in real-time, computing resources allocated to each queue also change over time. $\eta^{\mathrm{C}}_{i,j,j'}(t)=$

$$\left[\eta^{\mathrm{C}}_{i,j,j'}(t-1)+\lambda^{\mathrm{S}}_{i,j,j'}(t)-\frac{f^{\mathrm{C}}_{j'}\tau}{\lambda^{\mathrm{C}}_{i,j,j'}(t)b^{\mathrm{C}}_{j'}(t)}-\omega^{\mathrm{C}}_{i,j,j'}(t)\right]^{+}. \quad (8)$$

Let $l^{\mathrm{E}}_{i,j,j'}(t)\in\mathcal{T}$ as the time slot during which the offloaded task $k_{i,j,j'}(t)$ is either processed or dropped by neighboring EN j, and defined $\hat{l}^{\mathrm{E}}_{i,j,j'}(t)$ as the time slot at which the execution of task starts, we have:

$$\hat{l}^{\mathrm{E}}_{i,j,j'}(t)=\max\{t,\max_{t'\in\{0,1,\dots,t-1\}} l^{\mathrm{E}}_{i,j,j'}(t')+1\}, \quad (9)$$

where $l^{\mathrm{E}}_{i,j,j'}(0)=0$. Therefore, $l^{\mathrm{E}}_{i,j,j'}(t)$ is the time slot that satisfies the following constraints.

$$\sum_{t'=\hat{l}^{E}_{i,j,j'}(t)}^{l^{E}_{i,j,j'}(t)}\frac{f^{E}_{j'}/\lambda^{\mathrm{C}}_{i,j,j'}(t)}{b^{\mathrm{C}}_{j'}(t')}\geq\lambda^{\mathrm{S}}_{i,j,j'}(t)>\sum_{t'=\hat{l}^{E}_{i,j,j'}(t)}^{l^{E}_{i,j,j'}(t)}\frac{f^{E}_{j'}/\lambda^{\mathrm{C}}_{i,j,j'}(t)}{b^{\mathrm{C}}_{j'}(t')} \quad (10)$$

Problem Formulation: We define the task offloading as a cost minimization problem $\mathcal{P}$, which is affected by task exucation letancy and task drop penaly. The aim is to find the optimal policy $\boldsymbol{\pi}^{*}$ which minimizes the long-term cost, $\tilde{\mathcal{P}}:\boldsymbol{\pi}^{*}=:$

$$\min_{\boldsymbol{\pi}^{*}}\mathbb{E}\left[\gamma^{t}\left(\sum_{i\in\mathcal{I}}\sum_{t\in\mathcal{T}}\mathcal{C}_i(t)+\sum_{j\in\mathcal{J}}\sum_{i\in\mathcal{I}}\sum_{t\in\mathcal{T}}\mathcal{C}_{i,j}(t)\right)\middle|\boldsymbol{\pi}^{*}\right], \quad (11)$$

where $\gamma\in(0,1]$ is a discount factor and determines the balance between instant cost and long-term cost. The expectation $\mathbb{E}[\cdot]$ is taken into consideration of the time-varying system environments. Solving the optimization problem is particularly challenging due to the dynamic nature of the network. To address this challenge, we decouple the original problem ($\tilde{\mathcal{P}}$) into sub-problems (P1, P2) by decomposing the total objective into of different layers of offloading.

$$\text{P1:}\ \ \pi^{*}_{i}=\arg\ \min_{\pi_i}\mathbb{E}\left[\sum_{t\in\mathcal{T}}\gamma^{t-1}\mathcal{C}_i(t)\middle|\pi_i\right], \quad (12)$$

and

$$\text{P2:}\ \ \pi^{*}_{j}=\arg\ \min_{\pi_j}\mathbb{E}\left[\sum_{i\in\mathcal{I}}\sum_{t\in\mathcal{T}}\gamma^{t-1}\mathcal{C}_{i,j}(t)\middle|\pi_j\right]. \quad (13)$$

Policy for each agent is a mapping from its state to its corresponding action, denoted by i.e., $\pi_i:\mathcal{S}_i\rightarrow\mathcal{A}_i$ and $\pi_j:\mathcal{S}_j\rightarrow\mathcal{A}_j$. Especially, MD i and EN j determine the action $\boldsymbol{a}_i(t)\in\mathcal{A}_i$ and $\boldsymbol{a}_j(t)\in\mathcal{A}_j$, according to policy π_i given the observed environment state $\boldsymbol{s}_i(t)\in\mathcal{S}$.

IV. Multi-Agent DRL-Based Offloading Algorithm

Considering the coupled relation among the formulated problems, we first model the resource management problems as a decentralized partially observable markov decision process (Des-POMDP), where each MDs and ENs acts as an agent to learn the resource management scheme. Then, we introduce hierarchical multi-agent DRL-based offloading algorithm to learn the mapping between each state-action pair and their long-term cost to solve the corresponding formulated problems.

A. Problem Transformation

We transform the formulated problems into a markov game for $\mathcal{I}+\mathcal{J}$ agents as $\boldsymbol{\mathcal{S}}$, $\boldsymbol{\mathcal{O}}$, and $\boldsymbol{\mathcal{A}}$ for sets of states, observations, and actions, respectively. The state set $\boldsymbol{\mathcal{S}}$ describes the possible configurations of MDs and ENS, including the time-varying arrived tasks, queue's infomation, and historical data. $\mathcal{O}_i$ and $\mathcal{O}_j$ are observation spaces for the MD i agent and EN j agent, respectively, and the observation of each agent at time slot t is a part of the current state, $\mathcal{S}(t)\in\boldsymbol{\mathcal{S}}$. $\mathcal{A}_i$ and $\mathcal{A}_j$ are action spaces for the MD i and EN j. For each given state $\mathcal{S}(t)$, the MD agent and EN agent use the policies, $\pi_i:\mathcal{S}\rightarrow\mathcal{A}_i$ and $\pi_j:\mathcal{S}\rightarrow\mathcal{A}_j$, to choose an action from their action spaces according to their observations corresponding to $\mathcal{S}(t)\in\boldsymbol{\mathcal{S}}$.

Environment State: Based on the introduced system model and according to the offloading problems formulated for the MEC, the environment state at time slot t can define as $\mathcal{S}(t)$:

$$\mathcal{S}(t)=\big(k_i(t),\delta^{\mathrm{C}}_i(t),\delta^{\mathrm{T}}_i(t),k^{\mathrm{E}}_{i,j}(t),\eta^{\mathrm{E}}_{i,j}(t),\mathcal{H}(t)\big), \quad (14)$$

where matrix $\mathcal{H}(t)$ indicating the historical data of number of active queues $b^{\mathrm{C}}_{j}(t)$ for all ENs. This data is recorded over t^{s} time slots, ranging from $t-t^{\mathrm{s}}$ to $t-1$, in $t^{\mathrm{s}}\times J$ matrix.

Observation: Given the lack of information exchange among different agents, the observations made by MD i at time slot t are represented by $o_i(t)$ as:

$$o_i(t)=\big(k_i(t),\delta^{\mathrm{C}}_i(t),\delta^{\mathrm{C}}_i(t)\big). \quad (15)$$

And, also the EN j observations at time slot t, $o_j(t)$ is

$$\begin{aligned}o_j(t)=\Big(&k^{\mathrm{E}}_{1,j}(t),k^{\mathrm{E}}_{2,j}(t),...,k^{\mathrm{E}}_{I,j}(t),\eta^{\mathrm{E}}_{1,j}(t),\eta^{\mathrm{E}}_{2,j}(t),...\eta^{\mathrm{E}}_{I,j}(t),\\&b^{\mathrm{C}}_{1}(t-1),b^{\mathrm{C}}_{2}(t-1),...,b^{\mathrm{C}}_{J}(t-1),b^{\mathrm{C}}_{1}(t-2),b^{\mathrm{C}}_{2}(t-2),...,\\&b^{\mathrm{C}}_{J}(t-2),...,b^{\mathrm{C}}_{1}(t-t^{\mathrm{s}}),b^{\mathrm{C}}_{2}(t-t^{\mathrm{s}}),...,b^{\mathrm{C}}_{J}(t-t^{\mathrm{s}})\Big). \quad (16)\end{aligned}$$

Action: According to the current policy, π_i or π_j, and the corresponding observation, each MD or EN chooses an action $a_i(t)$ or $a_j(t)$ from its action space $\mathcal{A}_i(t)$ and $\mathcal{A}_j(t)$, respectively. The action of the MD i can be given by

$$a_i(t)=x_i(t). \quad (17)$$

The actions of EN j involves two decisions, $y_{i,j}(t)$ denote the offloading decision and offloading target for task arrived from MD $i\in\mathcal{I}$ in time slot t respectively. The EN's action j $a_j(t)\in\mathcal{A}_i$ in time slot t can be expressed as $a_j(t)=$

$$\big(y_{1,j}(t),y_{2,j}(t),...,y_{I,j}(t),\psi_{1,j}(t)),\psi_{2,j}(t),...,\psi_{I,j}(t)\big) \quad (18)$$

Cost Function: This system model hopes that each task will be completed within the maximum allowed duration; otherwise it will be penalized, which is typically considerably higher than the execution or transmission latency of the task. There is the cost $\mathcal{C}_i(t)$, $\mathcal{C}_j(t)$ associated with tasks for MD i and EN j at time t, given by

$$\mathcal{C}_i(t)=\begin{cases}l^{\mathrm{C}}_i(t)-t & \text{if}\quad 0<(1-x_i(t))\,l^{\mathrm{C}}_i(t)<\lambda^{d}_i(t)\\ \sum\limits_{\mathcal{J}}l^{\mathrm{E}}_{i,j}(t)-t & \text{if}\quad 0<x_i(t)\sum\limits_{\mathcal{J}}l^{\mathrm{E}}_{i,j}(t)<\lambda^{d}_i(t)\\ \Omega_i & \text{Otherwise,}\end{cases} \quad (19)$$

and

$$\mathcal{C}_{i,j}(t)=\begin{cases} l^{\mathrm{E}}_{i,j}(t)-t & \text{if } 0<(1-y_{i,j}(t))\, l^{\mathrm{E}}_{i,j}(t)<\lambda^d_i(t) \\ \sum\limits_{\mathcal{J}'} l^{\mathrm{E}}_{i,j,j'}(t)-t & \text{if } 0<y_{i,j}(t)\sum\limits_{\mathcal{J}} l^{\mathrm{E}}_{i,j}(t)<\lambda^d_i(t) \\ \Omega_j & \text{Otherwise.} \end{cases} \tag{20}$$

B. Agent Network Structure and Training Algorithm

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[2] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[3] L. Huang, S. Bi, and Y.-J. A. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Trans Mob Comput*, vol. 19, no. 11, pp. 2581–2593, Jul 2019.

[4] N. Zhao, Y.-C. Liang, D. Niyato, Y. Pei, M. Wu, and Y. Jiang, "Deep reinforcement learning for user association and resource allocation in heterogeneous cellular networks," *IEEE Trans. Wirel. Commun.*, vol. 18, no. 11, pp. 5141–5152, Aug 2019.

[5] M. Tang and V. W. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Trans Mob Comput*, vol. 21, no. 6, pp. 1985–1997, Nov 2020.

[6] H. Zhou, K. Jiang, X. Liu, X. Li, and V. C. Leung, "Deep reinforcement learning for energy-efficient computation offloading in mobile-edge computing," *IEEE Internet Things J.*, vol. 9, no. 2, pp. 1517–1530, Jun 2021.

[7] C. Sun, X. Li, C. Wang, Q. He, X. Wang, and V. C. Leung, "Hierarchical deep reinforcement learning for joint service caching and computation offloading in mobile edge-cloud computing," *accepted for publication in IEEE Trans. Services Computing*, 2024.

[8] I. Rahmati, H. Shah-Mansouri, and A. Movaghar, "Qeco: A qoe-oriented computation offloading algorithm based on deep reinforcement learning for mobile edge computing," *IEEE Trans. Netw. Sci. Eng.*, 2024.

[9] P. Hernandez-Leal, M. Kaisers, T. Baarslag, and E. M. De Cote, "A survey of learning in multiagent environments: Dealing with non-stationarity," *arXiv preprint arXiv:1707.09183*, 2017.

[10] J. Yang, Q. Yuan, S. Chen, H. He, X. Jiang, and X. Tan, "Cooperative task offloading for mobile edge computing based on multi-agent deep reinforcement learning," *IEEE Transactions on Network and Service Management*, vol. 20, no. 3, pp. 3205–3219, 2023.

[11] 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 Transactions on Mobile Computing*, vol. 22, no. 4, Apr 2021.

[12] H. Gao, X. Wang, W. Wei, A. Al-Dulaimi, and Y. Xu, "Com-ddpg: Task offloading based on multiagent reinforcement learning for information-communication-enhanced mobile edge computing in the internet of vehicles," *IEEE Transactions on Vehicular Technology*, 2023.

[13] S. Tan, B. Chen, D. Liu, J. Zhang, and L. Hanzo, "Communication-assisted multi-agent reinforcement learning improves task-offloading in uav-aided edge-computing networks," *IEEE Wireless Commun. Letters*, vol. 12, no. 12, pp. 2233–2237, Des 2023.

[14] M. S. Munir, N. H. Tran, W. Saad, and C. S. Hong, "Multi-agent meta-reinforcement learning for self-powered and sustainable edge computing systems," *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 3353–3374, 2021.

[15] A. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single-node case," *IEEE/ACM Trans. Netw*, vol. 1, no. 3, pp. 344–357, Jun 1993.