M^{3}OFF: Module-Compositional Model-Free Computation Offloading in Multi-Environment MEC

Tao Ren[*], Zheyuan Hu[†], Jianwei Niu[†‡§], Weikun Feng[†], Hang He[†]
[*] State Key Laboratory of Intelligent Game, Institute of Software Chinese Academy of Sciences, Beijing, China
[†] State Key Laboratory of Virtual Reality Technology and Systems, School of Computer Science and Engineering, Beihang University, Beijing, China
[‡] Zhongguancun Laboratory, Beijing, China
[§] Zhengzhou University Research Institute of Industrial Technology, School of Information Engineering, Zhengzhou University, Zhengzhou, China
Email: rentao22@iscas.ac.cn, {huzheyuan18, niujianwei, fengweikun98, hehang}@buaa.edu.cn

IEEE INFOCOM 2024 - IEEE Conference on Computer Communications | 979-8-3503-8350-8/24/$31.00 ©2024 IEEE | DOI: 10.1109/INFOCOM52122.2024.10621299

***Abstract*— Computation offloading is one of the key issues in mobile edge computing (MEC) that alleviates the tension between user equipment's limited capabilities and mobile application's high requirements. To achieve model-free computation offloading when reliable MEC dynamics are unavailable, deep reinforcement learning (DRL) has become a popular methodology. However, most existing DRL-based offloading approaches are developed for a single MEC environment, with invariant system bandwidth, edge capability, task types, etc., while realistic MEC scenarios tend to be of high diversity. Unfortunately, in multi-MEC environments, DRL-based offloading faces at least two challenges, learning inefficiency and interference of offloading experiences. To address the challenges, we propose a DRL-based Multi-environmental Module-compositional Model-free computation OFFloading (M^{3}OFF) framework. M^{3}OFF generates offloading policies using module composition instead of a single DRL network so that learning efficiency could be improved by reusing the same modules and learning interference could be reduced by composing different modules. Furthermore, we design multiple module composition-specific training methods for M^{3}OFF, including alternate modules-and-composer updates to improve training stability, loss-regularization to avoid module degeneration, and module-dropout to mitigate overfitting. Extensive experimental results on both simulation and testbed demonstrate that M^{3}OFF outperforms the performances of most state-of-the-arts in multi-MEC and reaches close to single-MEC.**

***Index Terms*—Mobile Edge Computing, Computation Offloading, Module Composition, Multiple Environments**

I. Introduction

The proliferation of smart user equipment (UE) has stimulated the extensive popularity of mobile applications, e.g., mobile payment, online gaming, smart healthcare, and augmented reality [1], which are usually computation-intensive and delay-critical. Whereas, UE is mostly equipped with limited computing resources and energy budget, resulting in a considerable gap between UE capability and application requirements. Assisted by the fast-evolving high-speed wireless communication technologies [2], mobile edge computing (MEC) [3] has been proposed as a promising paradigm to mitigate the gap, primarily by offloading UE's tasks to a nearby base station (BS) deployed with powerful computing resources for edge computing.

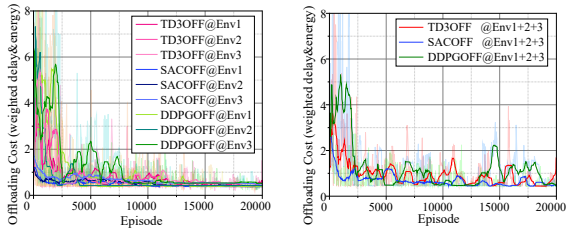
In MEC, one of the key issues is computation offloading that dynamically makes decisions (e.g., offloaded tasks, transmitting powers) according to time-varying MEC states (e.g., task requirements, energy budget, wireless conditions) [4] so that desirable quality of experience could be achieved for mobile applications. To make optimal decisions for computation offloading, traditional approaches are mainly developed based on mathematical programming, which depends heavily on reliable MEC system models [5], [6]. When system models are not available or reliable, approaches based on heuristic searching [7], [8], e.g., genetic algorithm and particle swarm optimization, become a possible way to achieve near-optimal computation offloading without knowing system models as a prior. Nonetheless, with the rapid development of wireless network speed, it becomes more and more difficult for these approaches to search for acceptable offloading solutions within a short time interval [9]. On account of this, there is a popular trend to develop model-free computation offloading approaches using reinforcement learning [10], or more often deep reinforcement learning (DRL) when the MEC states and offloading decisions are of high dimension.

Great efforts have been devoted to developing efficient DRL-based computation offloading approaches in various scenarios, e.g., multi-UE MEC [11], [12], multi-BS MEC [13], [14], vehicle-aided MEC [15] and satellite-integrated MEC [16], [17]. Whereas, most of these approaches only consider computation offloading in a single MEC environment (Env) with constant bandwidth, edge server (ES) capacity, task type, etc., while MEC Envs in realistic scenarios tend to be of high diversity. To the best of our knowledge, there has been little attention paid to multi-environmental computation offloading, triggering the motivation of this paper. However, when it comes to multi-environmental computation offloading, there will emerge at least two nontrivial issues. *The first is the experience-exploring and learning inefficiency*, since even only learning a DRL-based offloading policy in a single MEC Env could require a huge number of experiences [18], [19]. *The second is the experience-learning interference* during training DRL-based offloading

This work was supported in part by National Natural Science Foundation of China under Grant No. U23B2025, 62372027 and Zhejiang Provincial Natural Science Foundation of China under Grant LY22F020006. (Corresponding author: Jianwei Niu.)

Authorized licensed use limited to: Zhejiang University of Technology. Downloaded on December 13,2024 at 08:58:10 UTC from IEEE Xplore. Restrictions apply.

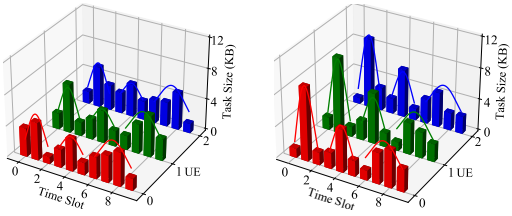
policies, where the gradients of exploring experiences from different MEC Envs could contradict each other, leading to significant performance degradation of computation offloading. To see the second issue obviously, we conduct a preliminary experiment to compare the performances of popular DRL-based offloading approaches [20]–[22] in single and multiple MEC Envs, as shown in Fig. 1. It shows that satisfactory computation offloading is more difficult to achieve in multiple MEC Envs than in single one, within acceptable episodes.



(a) Single MEC environment. (b) Multiple MEC environments.

Fig. 1. DRL-based computation offloading in single and multiple MEC Envs.

However, UE's offloading tasks in different MEC Envs could also share some common structures or patterns. Taking a real-world UE application dataset [23] as an example, we classify UE tasks in different Envs, and statistically analyze the rules of task sizes as shown in Fig. 2. The distribution of task sizes in one Env is found to be partially similar to that in another, e.g., the task size distributions of time slots 0-2 and 6-9 in 2(a) are similar to that of slots 3-5 and 6-9 in 2(b), respectively. Exploring and exploiting the common structures in different MEC Envs could be a possible way to improve multi-environmental offloading performance.



(a) Task sizes in MEC Env-A. (b) Task sizes in MEC Env-B.

Fig. 2. Data analysis of UE application tasks from a real-world dataset.

Motivated by this, we propose a Multi-environmental Module-compositional Model-free computation OFFloading (M^3OFF) framework. First, like most popular offloading works [11], [24], M^3OFF is developed based on DRL, and thus is model-free without the need to know MEC system models as prior. Second, M^3OFF is built by learning interaction experiences from multiple MEC Envs, apt to be more appropriate for applying in realistic MEC scenarios. Third, by making use of distinct modules in diverse tasks [25]–[27], module composition which generates offloading decisions by dynamically composing multi-layered modules, along with corresponding training methods, are proposed to improve learning efficiency and alleviate learning interference of M^3OFF in multiple MEC Envs. *A potential benefit of dynamic module composition is that common structures in different MEC Envs could be reused by the same module to improve learning efficiency, while different structures could be served by different modules to reduce learning interference.* Lastly, extensive experiments are conducted to verify the effectiveness of module composition and the superiority of M^3OFF against state-of-the-art DRL-based offloading approaches in various multi-Envs.

The main contributions are summarized as follows:

- We adopt the actor-critic DRL framework and employ multiple actors to interact with multiple MEC environments in parallel, achieving high experience-exploring efficiency and model-free computation offloading.
- We propose to develop the offloading actor using module composition, instead of a single neural network, which enables M^3OFF to improve experience-learning efficiency by reusing the same modules and reduce experience-learning interference by composing different modules.
- We further design multiple module composition-specific training methods for M^3OFF, including the alternate update of modules and composer to improve training stability, regularization of training loss to avoid module degeneration, and dropout of modules to mitigate overfitting.
- We conduct extensive ablation studies and visualization to evaluate the effectiveness of module composition in M^3OFF. Besides, we also demonstrate the superiority of M^3OFF over state-of-the-art DRL-based offloading approaches by more than 16.7% in both numerical simulation and real testbed.

II. System Model and Problem Formulation

Since the focus of this paper is to investigate multi-environmental computation offloading, we consider a typical MEC consisting of one BS (highly connected to an ES with computing frequency f^{ES}) and U UEs (denoted as $\mathcal{U}=\{1,2,...,U\}$ with each computing frequency being f^{UE}), as shown in Fig. 3. The system bandwidth is B MHz, and the system time is divided into N equal slots (each of which is ϵ s). Each UE $u\in\mathcal{U}$ generates a computation task $J_n^u=\{d_n^u, c, \tau\}$ at the beginning of slot n, where d_n^u denotes the input data size of J_n^u, c is the required CPU cycles to compute one bit of J_n^u, and τ is the maximum tolerable delay of J_n^u. Similar to [24], [28], J_n^u is assumed to be indivisible, and a binary variable $b_n^u\in\{0,1\}$ is used to denote whether J_n^u is executed locally on UE u (when $b_n^u=0$) or offloaded to ES for edge computing (when $b_n^u=1$). In addition, each UE u is equipped with a battery with energy budget e^{UE}, consumed by u during local computing or wireless transmitting of J_n^u in each slot n, and the available budget of u at the beginning of slot n is denoted by e_n^u. Both the UEs and ES are deployed with task queues to avoid task drops, with q_n^u and q_n^{ES} being the queue size of UE u and ES at the beginning of slot n, respectively.

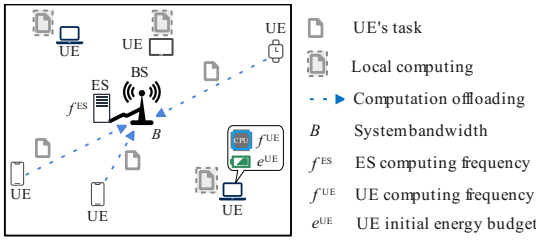


Fig. 3. A MEC network with one BS and multiple UEs.

Authorized licensed use limited to: Zhejiang University of Technology. Downloaded on December 13,2024 at 08:58:10 UTC from IEEE Xplore. Restrictions apply.

A. Communication Model

Similar to [29], [30], we adopt a block fading model to represent the channel gain from UE u to BS in slot n,

$$g_n^u = |h_n^u|^2 \beta_n^u, \tag{1}$$

where $|\cdot|$ is the mod function, h_n^u is the small-scale fading that could be modeled as a first-order Gaussian-Markov process[1] according to the Jakes' model [31], and β_n^u is the large-scale fading including both path-loss and log-normal shadowings. According to the LTE standard, β_n^u is modeled as

$$\beta_n^u = -120.9 - 37.6 \log_{10} \iota_n^u + 10 \log_{10} z, \tag{2}$$

where ι_n^u is the distance between UE u and BS in slot n, and z is a log-normal random variable obeying $\ln z \sim \mathcal{N}(0, \sigma_z^2)$.

The signal-to-interference-plus-noise ratio γ_n^u of the offloading link from UE u to BS in slot n, can be represented by

$$\gamma_n^u = \frac{g_n^u p_n^u}{\sum_{u' \neq u} g_n^{u'} p_n^{u'} + \sigma_\gamma^2}, \tag{3}$$

where p_n^u is the wireless transmitting power of UE u in slot n that should not exceed maximum UE transmitting power p^{UE}, and σ_γ^2 is the additive white Gaussian noise. Therefore, the transmitting rate from UE u to BS in slot n can be given by

$$\Gamma_n^u = B \log_2(1 + \gamma_n^u). \tag{4}$$

B. Edge Computing

If $b_n^u = 1$, then J_n^u will be offloaded to ES for edge computing. In this case, the task latency $t_n^{u,\text{edge}}$ includes the transmitting time $t_n^{u,\text{tran}}$, the queuing time $t_n^{u,\text{ESqueue}}$ in ES and the executing time $t_n^{u,\text{ESexe}}$ in ES, which can be calculated as

$$t_n^{u,\text{tran}} = \frac{d_n^u}{\Gamma_n^u}, \quad t_n^{u,\text{ESqueue}} = \frac{d_n^{\text{front}} c}{f^{\text{ES}}}, \quad t_n^{u,\text{ESexe}} = \frac{d_n^u c}{f^{\text{ES}}}, \tag{5}$$

respectively, where d_n^{front} is the sizes of tasks in front of J_n^u waiting for ES executing. Here, $d_n^{\text{front}} = q_n^{\text{ES}} + \sum_{u' \neq u} \mathbb{1}(t_n^{u',\text{tran}} < t_n^{u,\text{tran}}) d_n^{u'}$, including the queue size q_n^{ES} at the start of slot n and the data sizes of tasks finishing offloading before J_n^u.

Hence, we can get the task latency $t_n^{u,\text{edge}}$ and UE's energy consumption $e_n^{u,\text{edge}}$ for edge computing as follows,

$$t_n^{u,\text{edge}} = t_n^{u,\text{tran}} + t_n^{u,\text{ESqueue}} + t_n^{u,\text{ESexe}}, \tag{6}$$

$$e_n^{u,\text{edge}} = p_n^u t_n^{u,\text{tran}}, \tag{7}$$

where the UE's energy consumption during idle-waiting is omitted similar to [32], [33].

C. Local Computing

If $b_n^u = 0$, then J_n^u will be executed locally in UE u. Similarly, the task latency $t_n^{u,\text{loc}}$ includes the queuing time $t_n^{u,\text{UEqueue}}$ and executing time $t_n^{u,\text{UEexe}}$ in the UE, calculated as

$$t_n^{u,\text{UEqueue}} = \frac{q_n^u c}{f^{\text{UE}}}, \quad t_n^{u,\text{UEexe}} = \frac{d_n^u c}{f^{\text{UE}}}, \tag{8}$$

[1]The fading detail, which could be found in [29], is omitted for brevity.

respectively. Therefore, we can get the task latency $t_n^{u,\text{loc}}$ as

$$t_n^{u,\text{loc}} = t_n^{u,\text{UEqueue}} + t_n^{u,\text{UEexe}}, \tag{9}$$

and the corresponding energy consumption $e_n^{u,\text{loc}}$ as

$$e_n^{u,\text{loc}} = \xi (f^{\text{UE}})^2 d_n^u c, \tag{10}$$

where ξ denotes the energy efficiency coefficient depending on UE's chip architecture [34].

With the above models, the task latency t_n^u of J_n^u and energy consumption e_n^u of UE u in slot n could be calculated as

$$t_n^u = (1 - b_n^u) t_n^{u,\text{loc}} + b_n^u t_n^{u,\text{edge}}, \tag{11}$$

$$e_n^u = (1 - b_n^u) e_n^{u,\text{loc}} + b_n^u e_n^{u,\text{edge}}. \tag{12}$$

D. System Variables

System variables could be divided into two types: the time-varying internal variable $\nu_n^{\text{in}} = \{(e_n^u, q_n^u, q_n^{\text{ES}}) \,|\, u \in \mathcal{U}\}$ and time-invariant external (environmental) variable $\nu^{\text{env}} = \{P^{\text{task}}, B, f^{\text{UE}}, f^{\text{ES}}\}$, where P^{task} is the probability distribution of the task input data size, i.e., $d_n^u \sim P^{\text{task}}$. (1) For ν^{env}, its value represents a type of MEC Env, so we use $\mathscr{V} = \{\nu_1^{\text{env}}, \nu_2^{\text{env}}, ..., \nu_V^{\text{env}}\}$ to denote multiple types of Envs with V being the number of types. (2) For ν_n^{in}, its elements are time-dependent and transit over slots as follows,

$$e_{n+1}^u = e_n^u - \left[(1 - b_n^u) e_n^{u,\text{loc}} + b_n^u e_n^{u,\text{edge}}\right], \tag{13}$$

$$q_{n+1}^u = \max\{0,\ q_n^u + d_n^u - \frac{f^{\text{UE}} \epsilon}{c}\}, \tag{14}$$

$$q_{n+1}^{\text{ES}} = \max\{0,\ q_n^{\text{ES}} + \sum\nolimits_{u=1}^{U} b_n^u d_n^u - \frac{f^{\text{ES}} \epsilon}{c}\}. \tag{15}$$

E. Problem Formulation

Without losing generality, we define the objective of the computation offloading problem as minimizing the weighted cost of average task latency t_n and UE energy consumption e_n over multiple slots [35], [36], which can be expressed as

$$C_n^{\nu^{\text{env}}} = \eta t_n + (1 - \eta) e_n, \tag{16}$$

where $t_n = 1/U \sum_{u=1}^{U} t_n^u$ and $e_n = 1/U \sum_{u=1}^{U} e_n^u$, and the superscript ν^{env} in t_n and e_n are omitted for simplicity. Hence, taking $a_n^u = \{b_n^u, p_n^u\}$ as *the optimizing variable*, the optimization problem in ν^{env} could be formulated as

$$\text{P1}: \quad \min_{a_n^u} \quad C^{\nu^{\text{env}}} = \frac{1}{N} \sum_{n=1}^{N} C_n^{\nu^{\text{env}}} \tag{17}$$

$$\text{s.t.} \quad C1:\ b_n^u \in \{0, 1\}, \tag{18}$$

$$C2:\ p_n^u \leq p^{\text{UE}}, \tag{19}$$

$$C3:\ t_n^u \leq \tau, \tag{20}$$

$$C4:\ e_n^u \geq 0. \tag{21}$$

In multiple MEC Envs $\mathscr{V}$, the problem becomes

$$\text{P2}: \quad \min_{a_n^u} \ C^{\mathscr{V}} = \mathbb{E}_{\nu^{\text{env}} \sim \mathscr{V}} C^{\nu^{\text{env}}}, \tag{22}$$

which is subject to the same constraints from Eqs. (18) to (21).

Authorized licensed use limited to: Zhejiang University of Technology. Downloaded on December 13,2024 at 08:58:10 UTC from IEEE Xplore. Restrictions apply.

F. Problem Transformation

It can be observed from Eqs. (13) to (15) that the decision of the current a_n^u depends tightly on the internal variables caused by previous $a_{n'}^u$ where $n' < n$. Therefore, we transform P1 into an MDP, adopted widely in MEC [37]–[39] to solve sequential decision-making problems. We define a tuple $< S, A, Pr, R, \gamma >$ to represent the MDP, where: (1) S is the state space with each state $s_n \in S$ in slot n being defined as $s_n = \{(J_n^u, B, f^{\text{UE}}, f^{\text{ES}}, e_n^u, q_n^u, q_n^{\text{ES}}) | u \in \mathscr{U}\}$, (2) A is the action space with each action $a_n \in A$ in slot n being defined as $a_n = \{a_n^u | u \in \mathscr{U}\}$, (3) $Pr(s_{n+1}|s_n, a_n)$ is the state transition probability determining the transiting probability from s_n to s_{n+1} undertaking a_n, and is not known as a prior, (4) $R(s_n, a_n)$ is a reward function generating an immediate reward[2] in each slot n according to the impacts of taking a_n in s_n on the optimization objective and constraints, (5) γ is a discount factor satisfying $\gamma \in (0, 1]$ and determining the impact of future rewards.

With the MDP, we can transform P1 into P3 as follows,

$$\text{P3}: \ \underset{\pi}{\operatorname{argmax}} \, \mathbb{E}_{a_n \sim \pi, s_n \sim \nu^{\text{env}}} \frac{1}{N} \sum_{n=1}^{N} \gamma^{n-1} R(s_n, a_n), \tag{23}$$

which aims to obtain a policy π that maximizes expected long-term rewards in an individual MEC Env ν^{env}. Correspondingly, in multiple MEC Envs $\mathscr{V}$, the problem P2 can be transformed into P4 as follows,

$$\text{P4}: \underset{\pi}{\operatorname{argmax}} \, \mathbb{E}_{\nu^{\text{env}} \sim \mathscr{V}} \mathbb{E}_{a_n \sim \pi, s_n \sim \nu^{\text{env}}} \frac{1}{N} \sum_{n=1}^{N} \gamma^{n-1} R(s_n, a_n), \tag{24}$$

aiming to get a policy π maximizing expected long-term rewards in multiple MEC Envs $\mathscr{V}$, rather than a single one.

Without knowing the system dynamics $Pr(s_{n+1}|s_n, a_n)$ as a prior, we can adopt DRL to solve P4 by learning an offloading policy using interaction experiences sampled from multiple MEC Envs $\mathscr{V}$. However, this instinctive method could be rather challenging due to the following reasons: (1) The learning process could be very experience-consuming, since only learning DRL-based policies in a single MEC Env usually requires a large amount of experiences [18], [19]; (2) Experiences from different MEC Envs could interfere with each other during learning a single DRL-based offloading model. *These are also the reasons for this paper to study multi-environmental computation offloading and propose a Multi-environmental Module-compositional Model-free computation OFFloading framework — M^{3}OFF, that can improve learning efficiency by reusing offloading modules and reduce interference by composing different modules* as detailed in the next section.

III. Multi-Environmental Module-Compositional Computation Offloading

In this section, we will describe the framework, module composition and training methods of M^{3}OFF in detail.

[2] $R(s_n, a_n)$ is defined as $\mathbb{1}(\cdot)(C_n^{\nu^{\text{env}}})^{-1} + (1 - \mathbb{1}(\cdot)) \sum w^j p_j$, where $\mathbb{1}(\cdot)$ indicates whether all constraints are satisfied, w^j is the penalty-weight of the constraint Cj, and p_j is a penalty depending on the degree of violating Cj.

A. DRL-based Multi-Environmental Framework

On the whole, M^{3}OFF is a model-free DRL-based computation offloading scheme, whose framework is shown in Fig. 4, achieving long-term reward-maximizing computation offloading using the policy learned by exploring and exploiting experiences from MEC Envs $\mathscr{V}$. The DRL agent (in the top of Fig. 4) is built based on popular actor-critic architectures, where the actor is responsible for interacting with the MEC environment while the critic evaluates the impacts of actor's actions on the long-term reward. During training, to increase experience exploring efficiency, we simultaneously activate V actors of the DRL agent to interact with V MEC Envs (at the bottom of Fig. 4). The interaction experience (s_n, a_n, r_n, s_{n+1}) [3], along with the corresponding MEC environmental variable ν^{env}, are stored in an experience buffer, which are then sampled as a batch $\mathscr{E}$ to update the parameters of the actor and critic (usually implemented as neural networks and updated using back-propagation) under specific loss functions detailed below. After training, the obtained actor could generate offloading actions in the multiple MEC Envs $\mathscr{V}$.

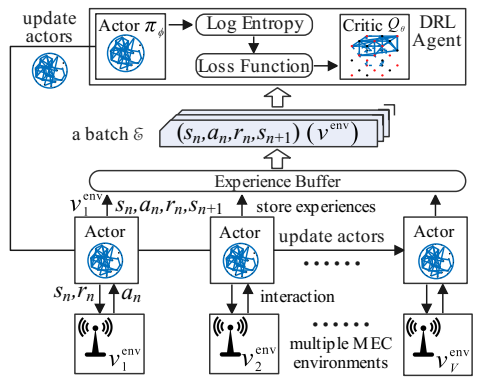


Fig. 4. The framework overview of M^{3}OFF.

Supposing the actor π_ϕ and critic Q_θ are parameterized by ϕ and θ, respectively, the long-term reward generated by Q_θ could be represented as a Q value function,

$$Q_\theta(s_n, a_n) = \mathbb{E}_{\nu^{\text{env}} \sim \mathscr{V}} \mathbb{E}_{a_n \sim \pi_\phi, s_n \sim \nu^{\text{env}}} \frac{1}{N} \sum_{n=1}^{N} \gamma^{n-1} R(s_n, a_n). \tag{25}$$

In addition, we adopt soft actor-critic [40] to improve the exploring capability of the actor by introducing an action-entropy into the optimizing objective function. The policy π_ϕ is seemed as a multi-dimension Gaussian distribution $\mathscr{N}_\phi(\boldsymbol{\mu}, \boldsymbol{\sigma})$, and the mean μ and co-variance σ of each element in a_n is approximated by π_ϕ, i.e., $\pi_\phi(a_n|s_n)$ denotes the Gaussian probability distribution of choosing a_n in s_n.

The parameters ϕ and θ can be optimized by minimizing

$$J_\pi(\phi) = \mathbb{E}_{s_n \sim \mathscr{E}} \left[\mathrm{D}_{\text{KL}} \left(\pi_\phi(\cdot|s_n) \middle\| \frac{\exp(Q_\theta(s_n, \cdot))}{Z_\theta(s_n)} \right) \right], \tag{26}$$

$$J_Q(\theta) = \mathbb{E}_{s_n, a_n \sim \mathscr{E}} \left[\frac{1}{2} \Big(Q_\theta(s_n, a_n) - \big(r_n + \gamma \mathbb{E}_{s_{n+1} \sim Pr} V_{\bar{\psi}}(s_{n+1}) \big) \Big)^2 \right], \tag{27}$$

[3] r_n is the output of the reward function $R(s_n, a_n)$ defined previously.

Authorized licensed use limited to: Zhejiang University of Technology. Downloaded on December 13,2024 at 08:58:10 UTC from IEEE Xplore. Restrictions apply.

where $\mathrm{D_{KL}}$ denotes the KL-divergence, $Z_\theta(s_n)$ is the partition function to normalize $\exp(Q_\theta(s_n,\cdot))$, and $V_\psi(s_n) = \mathbb{E}_{a_n\sim\pi}[Q_\theta(s_n,a_n) - \log\pi(a_n|s_n)]$ is a separate soft state value function parameterized by ψ (with $\bar{\psi}$ denoting the target network of V_ψ used for function stability). The parameter ψ can be optimized by minimizing

$$J_V(\psi)=\mathbb{E}_{s_n\sim\mathscr{E}}\Big[\frac{1}{2}\Big(V_\psi(s_n)-\mathbb{E}_{a_n\sim\pi}\big(Q_\theta(s_n,a_n)-\log\pi(a_n|s_n)\big)\Big)^2\Big]. \tag{28}$$

B. Multi-Environmental Module-Compositional Actor

In the above framework, the actors π_ϕ need to interact with multiple MEC Envs $\mathscr{V}$, and the large amount of interaction experiences coming from different ν^{env}s put forward severe challenges to learn a single neural network actor π_ϕ, due to not only the training efficiency but also the gradient interference of the actor π_ϕ during back-propagation. For example, if the gradient of a parameter of ϕ derived from the experiences from a MEC Env ν^{env} wants to guide the parameter to update toward one direction while the gradient from another MEC Env $\nu^{\text{env}'}$ toward the opposite direction, the actor π_ϕ will suffer from obvious performance degradation. Especially, when the number V is large, the interference will occur more severely.

Facing the issue, we propose to build the offloading actor π_ϕ (i.e., $\mathcal{N}_\phi(\boldsymbol{\mu},\boldsymbol{\sigma})$) using multi-layered modules (MLMs), instead of a single neural network, as shown in Fig. 5. The potential benefits could include not only the improvement of the learning efficiency of π_ϕ by reusing the same module for similar offloading actions in different ν^{env} but also the reduction of learning interference among different ν^{env} by composing different modules. *To enable reusing same modules and composing different modules, we additionally propose to use a dynamic-composer (DC) to dynamically decide which module to reuse or not reuse for different ν^{env}.* Hence, the actor π_ϕ (i.e., $\mathcal{N}_\phi(\boldsymbol{\mu},\boldsymbol{\sigma})$) in previous section, is jointly produced here by MLMs (parameterized by ϕ) and DC (parameterized by φ), and becomes a module-compositional actor (MC-actor) $\mathcal{N}_{\phi,\varphi}(\boldsymbol{\mu},\boldsymbol{\sigma})$, as shown in Fig. 5.

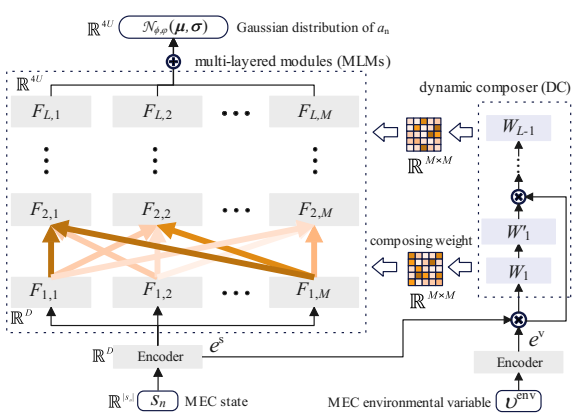


Fig. 5. The framework of the module-compositional actor (MC-actor).

*1) **Multi-Layered Modules**:* There are L layers of modules, with each layer containing M modules. Each module $F_{\ell,m}, m\in\{1,2,...M\}$ in the ℓ-th layer could be viewed as a function block with input dimension D_ℓ^{I} and output dimension D_ℓ^{O}, satisfying two conditions: First, $F_{\ell,m}, m\in\{1,2,...M\}$ in the same layer ℓ should be of the same dimension; Second, the output dimension D_ℓ^{O} of the layer ℓ should equal to the input dimension $D_{\ell+1}^{\mathrm{I}}$ of the next layer $\ell+1$. For the first layer, the input dimension is equal to D, which is consistent with the dimension of the state embedding e^{s}. The embedding e^{s} is generated by an encoder, which takes the MEC state s_n as input. For the last layer, the output dimension is $4U$, equaling to the dimension of the Gaussian parameters $\mathcal{N}_{\phi,\varphi}(\boldsymbol{\mu},\boldsymbol{\sigma})$ representing the element-wise distribution of the offloading action a_n.

*2) **Dynamic Composer**:* There are $L-1$ layers in the dynamic composer, i.e., DC, and the output of each DC layer ℓ is denoted as $\rho_{i,j}^\ell$ with $i,j\in\{1,2,...,M\}$. The output $\rho_{i,j}^\ell$ of DC layer ℓ is responsible for dynamically composing the outputs of modules $F_{\ell,j}, j\in\{1,2,...M\}$ in layer ℓ to provide inputs for the modules $F_{\ell+1,i}, i\in\{1,2,...M\}$ in layer $\ell+1$. Taking the k-th input $I_{\ell+1,i}^k$ of $F_{\ell+1,i}$ as example, $I_{\ell+1,i}^k$ is produced by DC as follows,

$$I_{\ell+1,i}^k = \sum_{j=1}^{M}\big[\rho_{i,j}^\ell O_{\ell,j}^k\big], \tag{29}$$

where $O_{\ell,j}^k$ is the k-th output of the j-th module $F_{\ell,j}$ in layer ℓ. For the DC, the W_1 in the first layer takes the element-wise product (whose dimension is D) of the state embedding e^{s} and the environmental variable embedding e^{v} as input and generates $\rho_{i,j}^1$ as the output. Then, $\rho_{i,j}^1$ is input to W_1' to produce a D-dimensional output, which is further element-wisely multiplied by the element-wise product of e^{s} and e^{v} to produce a D-dimensional input for the next layer of DC. It is worth noting that the inputs of MLMs and DC are designed asymmetrically so that *the MLMs can focus on extracting reusable modules regardless of what the ν^{env} is and the DC can learn to compose appropriate modules according to ν^{env}.*

*3) **Implementation Considerations**:* Each module $F_{\ell,m}$ could be implemented using mathematical functions with different equations or neural networks with different hidden architectures, as long as their input and output dimensions satisfy the above two conditions about D_ℓ^{I} and D_ℓ^{O}. It is the same for implementing $W_\ell, W_\ell', \ell=\{1,2,...,L-1\}$ in DC. Since the framework of M^{3}OFF is based on DRL, we build MLMs' modules $F_{\ell,m}$ and DC's components W_ℓ, W_ℓ' using multi-layered perception (MLP), whose architectures will be detailed in the experiments. Similarly, the two encoders for extracting e^{s} and e^{v} from s_n and ν^{env}, respectively, are also implemented using MLP. Additionally, we adopt the categorical reparameterization method, Gumbel-Softmax [41], to address the back-propagation difficulty resulted from the non-differentiable sampling of a_n from $\mathcal{N}_{\phi,\varphi}(\boldsymbol{\mu},\boldsymbol{\sigma})$, so that the MC-actor which is differentiable itself as designed above could be incorporated with the M^{3}OFF framework to enable end-to-end training.

To this end, with ϕ and φ (whose subscripts ℓ,m are omitted for brevity) denoting the neural network parameters of MLMs (F^ϕ) and DC (W^φ and W'^φ), respectively, the loss function in Eq. (26) of the offloading policy π could be rewritten as

$$J_\pi(\phi,\varphi)=\mathbb{E}_{s_n\sim\mathscr{E}}\Big[\mathrm{D_{KL}}\Big(\pi_{\phi,\varphi}(\cdot|s_n)\Big|\Big|\frac{\exp(Q_\theta(s_n,\cdot))}{Z_\theta(s_n)}\Big)\Big]. \tag{30}$$

Authorized licensed use limited to: Zhejiang University of Technology. Downloaded on December 13,2024 at 08:58:10 UTC from IEEE Xplore. Restrictions apply.

C. Training Algorithm and Method

Giving the above description, we design the end-to-end training algorithm for M^{3}OFF, as shown in Algorithm 1. First, V MC-actors are started in parallel to interact with individual MEC Envs, i.e., $\Phi^v \rightleftarrows \nu_v^{\text{env}}$, in Lines 6-17, to improve experience-exploring efficiency. Second, both the parameters ϕ and φ of MLMs and DC are updated in Lines 19-21, using experiences sampled from $\mathscr{E}$. Third, the learned parameters ϕ and φ are periodically updated to V parallel MC-actors in Line 22.

Algorithm 1 The training algorithm of M^{3}OFF.

1: Init $\theta, \psi, \phi, \varphi$
2: Init target parameters $\bar{\psi} \leftarrow \psi$
3: Init experience buffer $\mathscr{E}$
4: Init V MC-actors $\Phi = \{(\phi^v, \varphi^v) | v = 1,2,...,V\}$ using ϕ, φ
5: Start V pairs of interaction: $\Phi^v \rightleftarrows \nu_v^{\text{env}}$, $v = 1, 2, ..., V$
6: **for** each $\Phi^v \rightleftarrows \nu_v^{\text{env}}$ **do in parallel**:
7: **repeat**
8: Initialize the MEC environment ν_v^{env}
9: **for** each slot n **do**
10: Observe s_n
11: Generate $\boldsymbol{\mu}, \boldsymbol{\sigma}$ by Φ^v using s_n and ν_v^{env}
12: sample a_n from $\mathscr{N}_{\phi,\varphi}(\boldsymbol{\mu}, \boldsymbol{\sigma})$
13: Perform a_n with ν_v^{env} and receive s_{n+1}, r_n
14: Store $(s_n, a_n, r_n, s_{n+1}, \nu_v^{\text{env}})$ into $\mathscr{E}$
15: **end for**
16: **until** true
17: **end for**
18: **repeat** training asynchronously:
19: Sample a batch of $(s_j, a_j, r_j, s_{j+1}, \nu_v^{\text{env}})$ from $\mathscr{E}$
20: Alternately update ϕ, φ based on $\nabla_\phi \pi, \nabla_\varphi \pi$ computed using sampled experiences according to Eq. (31)
21: Update θ, ψ based on $\nabla_\theta Q, \nabla_\psi V$ computed using sampled experiences according to Eqs. (27) and (28)
22: Periodically update Φ using ϕ and φ
23: Periodically update $\bar{\psi}$ using ψ
24: **until** convergence
25: Stop $\Phi^v \rightleftarrows \nu_v^{\text{env}}$, $v = 1, 2, ..., V$
26: **return** ϕ, φ

Moreover, on account of the adoption of module composition in M^{3}OFF, we design the following methods to further promote training and offloading performances of M^{3}OFF:

*1) **Alternate Update to Improve Training Stability**:* According to the description of the MC-actor in Section III-B, MLMs' parameter ϕ collaborates with DC's parameter φ to dynamically compose the offloading policy distribution $\mathscr{N}_{\phi,\varphi}(\boldsymbol{\mu}, \boldsymbol{\sigma})$, which is then used to sample a_n to interact with ν^{env} and generate experiences that are in turn used to update both ϕ and φ simultaneously. It means that either ϕ or φ affects the update of them both, hence, simultaneously updating ϕ and φ would be apt to bring unexpected impact on training stability of both ϕ and φ. In view of this, we propose to alternately update ϕ and φ to improve the training stability of M^{3}OFF, and set the alternate training periods of ϕ and φ as Γ_ϕ and Γ_φ, respectively, the ratio of which will be empirically investigated in the experiment.

*2) **Regularization to Avoid Module Degeneration**:* During the early training period, if one module is initialized and trained with higher expected return, it would be selected more often than other modules by DC and thus contribute more exploring experiences and receive more training. This type of exclusive selection and training of one module will make others degenerate, hurting the expression capability of MLMs and MC-actor. Therefore, we design a regularization term $\mathscr{R}$ and impose it in the loss function of MC-actor to avoid module degeneration as follows,

$$J_\pi(\phi, \varphi) = \mathbb{E}_{s_n \sim \mathscr{E}} \Big[\mathrm{D}_{\mathrm{KL}} \Big(\pi_{\phi,\varphi}(\cdot|s_n) \Big\| \frac{\exp(Q_\theta(s_n, \cdot))}{Z_\theta(s_n)} \Big) + \mathscr{R} \Big]. \tag{31}$$

$\mathscr{R}$ is designed by taking the long-term sum $G_j^\ell = \sum_{\mathscr{T}} \sum_{i=1}^{M} \rho_{i,j}^\ell$ of composing weights of each module j in layer ℓ into account, where $\mathscr{T}$ is the time horizon. With the expectation to prevent some module from being exclusively selected and trained, we define $\mathscr{R} = \max_{\forall \ell, j \neq j'} (G_j^\ell - G_{j'}^\ell)^2$ with the goal of minimizing the gap between any two modules j and j' in any layer ℓ.

*3) **Module Dropout to Mitigate Overfitting**:* Due to the adoption of module composition in M^{3}OFF, it becomes possible for the DC to make use of its flexible composing capability to compose highly local-optimal modules to overfit a complex offloading policy distribution, something like the Gaussian mixture models. To mitigate this issue, we adopt the dropout method that randomly drops modules with some probability. A vector $\{p_{\ell,m}^{\text{drop}} \,|\, \ell = 1, 2, ..., L, m = 1, 2, ..., M\}$ is maintained, each element of which is a Bernoulli random variable with the probability p^{drop} being 1. Hence, the calculation for the k-th input $I_{\ell+1,i}^k$ of module $F_{\ell+1,i}$ in Eq. (29) is rewritten as

$$I_{\ell+1,i}^k = \sum_{j=1}^{M} \big[\rho_{i,j}^\ell (p_{\ell,j}^{\text{drop}} O_{\ell,j}^k) \big]. \tag{32}$$

IV. Experiments

We evaluate the performance of M^{3}OFF in various MEC environments via both simulation and testbed.

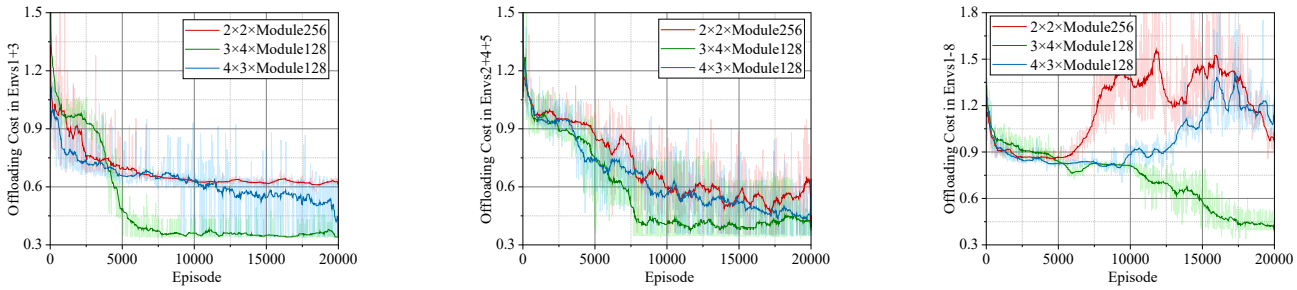
A. Experiment Settings

1) MEC and M^{3}OFF Settings: We build the MEC system and DRL-based offloading approaches using Python 3.6 and PyTorch 1.10. Primary parameters of the MEC system are listed in Table I. For DRL related parameters, the discount factor, learning rate, size of experience buffer and batch size are set as 0.98, 10^{-3}, 4000 and 64, respectively. For parameters of the training methods of M^{3}OFF, $\mathscr{T}$ and p^{drop} are set as 100 and 0.9, respectively.

TABLE I
Primary MEC Parameters.

Parameter	Description	Value
U	Number of UEs	12
N	Number of time slots	10
ϵ	Time slot length	0.5 s
c	CPU cycles per bit of data	900
τ	Maximum tolerable delay	0.9 s
p^{UE}	Maximum transmitting power	0.5 W
f^{UE}	UE's computing capacity	1 GHz
e^{UE}	UE's initial energy budget	7 J
ξ	Energy efficiency coefficient	10^{-27}
σ_γ^2	Additive white Gaussian noise	-100 dBm
η	The cost weight for task latency	0.5

Authorized licensed use limited to: Zhejiang University of Technology. Downloaded on December 13,2024 at 08:58:10 UTC from IEEE Xplore. Restrictions apply.



(a) Performance evaluation in Envs1+3. (b) Performance evaluation in Envs2+4+5. (c) Performance evaluation in Envs1-8.

Fig. 6. Performance evaluation on the module layers and sizes of M^{3}OFF in MEC Envs with different sizes ($|\mathscr{V}| = 2, 5, 8$).

2) Multi-Env Settings: We define 8 MEC environments Env1-Env8, as shown in Table II, by combining different $v^{\text{env}} = \{P^{\text{task}}, B, f^{\text{ES}}\}$ (given in Table III), with the task size distribution P^{task} set according to the real-world UE application data [23].

TABLE II
MULTIPLE MEC ENVIRONMENTS.

Env Setting							
Index	Slot 1-3	Slot 4-6	Slot 7-10	Index	Slot 1-3	Slot 4-6	Slot 7-10
Env1	v_5^{env}	v_7^{env}	v_9^{env}	Env5	v_2^{env}	v_6^{env}	v_{10}^{env}
Env2	v_2^{env}	v_4^{env}	v_{10}^{env}	Env6	v_1^{env}	v_2^{env}	v_3^{env}
Env3	v_4^{env}	v_5^{env}	v_9^{env}	Env7	v_3^{env}	v_5^{env}	v_6^{env}
Env4	v_4^{env}	v_8^{env}	v_{10}^{env}	Env8	v_7^{env}	v_5^{env}	v_2^{env}

TABLE III
VALUES OF v^{ENV} W.R.T. P^{TASK} (KB), B (MHz) AND f^{ES} (GHz).

Values of v^{env}							
No.	P^{task}	B	f^{ES}	No.	P^{task}	B	f^{ES}
v_1^{env}	$\mathcal{N}(500, 100)$	11	9	v_6^{env}	$\mathcal{N}(500, 100)$	8.5	7
v_2^{env}	$\mathcal{N}(400, 100)$	10.5	8	v_7^{env}	$\mathcal{N}(800, 100)$	9	8
v_3^{env}	$\mathcal{N}(300, 100)$	10.5	7	v_8^{env}	$\mathcal{N}(700, 100)$	10.5	6
v_4^{env}	$\mathcal{N}(700, 100)$	10.5	9	v_9^{env}	$\mathcal{N}(400, 100)$	9.5	9
v_5^{env}	$\mathcal{N}(600, 100)$	9.5	8	v_{10}^{env}	$\mathcal{N}(300, 100)$	10	8

3) Metrics and Baselines: We take the multi-Env offloading cost $C^{\mathscr{V}}$ in Eq. (22) as the evaluation metric. Baselines include state-of-the-art DRL-based offloading approaches in latest MEC works, namely, SACOFF (SAC-based offloading [22]), TD3OFF (TD3-based offloading [21]) and DDPGOFF (DDPG-based offloading [20]), each of which is built directly using one neural network as the actor to learn a multi-Env offloading policy. Moreover, we also take a synthetic baseline SINOFF for comparison, whose cost is obtained by averaging the costs of SAC individually trained in each *single* MEC Env.

B. Performance Evaluation of M^{3}OFF

We first investigate the impacts of different combinations of module layers and module sizes on the performance of M^{3}OFF in MEC Envs with distinct sizes (2 Envs, 3 Envs, 8 Envs), as shown in Fig. 6. The combinations include $2\times2\times$Module256 ($L = 2, M = 2$, 256×256 module), $3\times4\times$Module128 ($L = 3, M = 4$, 128×128 module) and $4\times3\times$Module128 ($L = 4, M = 3$, 128×128 module), with close numbers of neural-parameters. The offloading cost $C^{\mathscr{V}}$ of each combination is seen to increase gradually when $|\mathscr{V}|$ grows from 2 in Fig. 6(a) to 8 in Fig. 6(c). For instance, $3\times4\times$Module128 converges to about 0.376, 0.404 and 0.465 at the episode around 6000, 8000 and 17000, respectively, as shown in Figs. 6(a) to 6(c), suggesting that the offloading performance and learning difficulty are largely affected by the number of MEC Envs. It could also be noticed that $3\times4\times$Module128 could achieve more promising offloading performance and learning efficiency in any of the three different-size MEC Envs. Therefore, we will take $3\times4\times$Module128 for M^{3}OFF in the following part.

We then evaluate the impacts of different alternate update ratios $\Gamma_\phi/\Gamma_\varphi$ of MLMs and DC for M^{3}OFF in multi-MEC Envs (2 Envs, 3 Envs, 8 Envs), as shown in Table IV. As can be seen, a large ratio 4/1 achieves more promising training performance than other two ratios 1/4 and 5/5 in any of the three Envs, possibly due to the relatively larger number of parameters of MLMs than DC. Better performance may be possible by adjusting the ratio dynamically during the training of M^{3}OFF, which could be a future work. Hence, we take 4/1 as the alternate update ratio in the following part. In Fig. 7, we also evaluate the impacts of three learning-enhancing methods designed in Section III-C, i.e., alternate update, loss regularization and module dropout. M^{3}OFF+W/O, M^{3}OFF+AU, M^{3}OFF+LR, M^{3}OFF+MD and M^{3}OFF+ALL denote the M^{3}OFF without any method, with alternate update, with loss regularization, with module dropout, and with all three methods, respectively. The cost of M^{3}OFF+W/O is observed to be the highest, and M^{3}OFF+AU is lower than M^{3}OFF+LR and M^{3}OFF+MD. On the whole, M^{3}OFF+ALL is noticed to converge to the lowest offloading cost around 0.46, indicating the effectiveness of the designed methods. Hence, we will use M^{3}OFF+ALL in the following experiments.

TABLE IV
IMPACTS OF ALTERNATE UPDATE RATIOS OF MLMS AND DC FOR M^{3}OFF

$\Gamma_\phi/\Gamma_\varphi$	Envs1+3			Envs2+4+5			Envs1-8		
	1/4	1/1	4/1	1/4	1/1	4/1	1/4	1/1	4/1
Converging episode	9800	6700	**6050**	8900	11900	**7960**	\	\	**16900**
Converging cost	0.85	0.76	**0.38**	0.95	0.55	**0.40**	\	\	**0.46**

We further investigate the impacts of the size of Envs (i.e., $|\mathscr{V}|$) on the offloading performance of M^{3}OFF. We take 4

Authorized licensed use limited to: Zhejiang University of Technology. Downloaded on December 13,2024 at 08:58:10 UTC from IEEE Xplore. Restrictions apply.

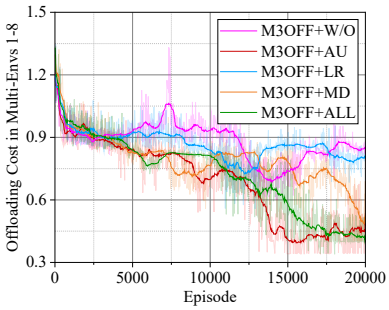


Fig. 7. Ablation study of three training methods.

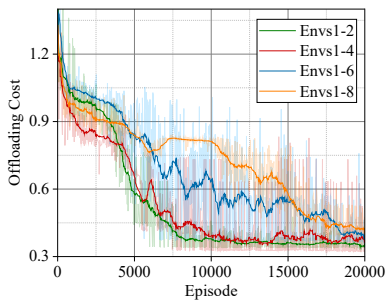


Fig. 8. Impacts of MEC Envs-sizes (i.e., $|\mathscr{V}|$).

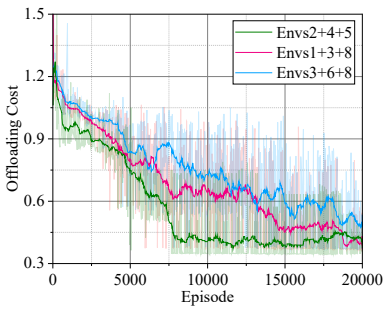
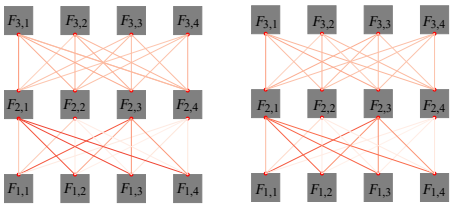


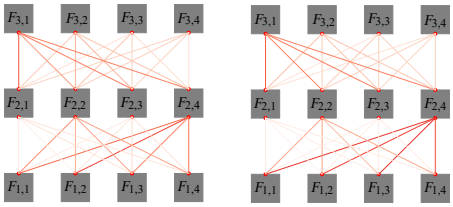
Fig. 9. Impacts of distinct MEC Envs.

types of sizes for evaluation, including Envs1-2 ($|\mathscr{V}| = 2$), Envs1-4 ($|\mathscr{V}| = 4$), Envs1-6 ($|\mathscr{V}| = 6$), Envs1-8 ($|\mathscr{V}| = 8$). As shown in Fig. 8, the offloading performance of M^{3}OFF is observed to rise gradually with the decrease of $|\mathscr{V}|$ from 8 to 2, and M^{3}OFF is found to converge fastest to the lowest cost 0.386 in Envs1-2 among 4 types of Env-sizes. It suggests the increase of Envs-size would bring more troubles for M^{3}OFF, since the more is the number of distinct Envs the greater efforts are required to made by M^{3}OFF to distinguish different Envs, select modules for reuse and avoid interference.

Additionally, we investigate the impacts of different combinations of Envs on M^{3}OFF when the Envs-sizes are fixed ($|\mathscr{V}| = 3$), as shown in Fig. 9. It can be seen that Envs2+4+5 achieves the lowest offloading cost around 0.41 while Envs3+6+8 the highest around 0.55. It indicates that even when the Envs-sizes are the same, the offloading difficulties could also be of large variety in different combinations of Envs due to the different similarities of Envs within a combination. The combination with higher similarities of Envs, like Envs2+4+5, could be easier for the reuse of modules in M^{3}OFF, thereby leading to higher offloading performance.



(a) Env1 ($v_5^{\text{env}}, v_7^{\text{env}}, v_9^{\text{env}}$). (b) Env3 ($v_4^{\text{env}}, v_5^{\text{env}}, v_9^{\text{env}}$).



(c) Env2 ($v_2^{\text{env}}, v_4^{\text{env}}, v_{10}^{\text{env}}$). (d) Env4 ($v_4^{\text{env}}, v_8^{\text{env}}, v_{10}^{\text{env}}$).

Fig. 10. Visualization of DC-output weights for dynamic module composition in different MEC environments.

Furthermore, to take an insight into the effect of the dynamic module composition on improving learning efficiency in multiple MEC Envs, we sample the weights $\rho_{i,j}^{\ell}$ of each M^{3}OFF's DC layer and visualize them in different MEC Envs, with deeper red lines representing larger composing weights as shown in Fig. 10. It can be noticed that the color of lines ($F_{2,1}-F_{1,2}$, $F_{1,3}$, $F_{1,4}$) and ($F_{2,3}-F_{1,1}$, $F_{1,2}$) in Fig. 10(a) for Env1 is deeper than others, which is also observed in Fig. 10(b) for Env3. Since Env1 and Env3 are highly similar to each other (with both v_5^{env} and v_9^{env}), it indicates the modules $F_{2,1}$, $F_{2,3}$, $F_{1,1}$-$F_{1,4}$ are highly reused in Env1 and Env3. Similar cases could be seen in Figs. 10(c) and 10(d) as well, where Env2 and Env4 also share similar Env settings (v_4^{env} and v_{10}^{env}).

We also cluster and visualize the DC-output weights $\rho_{i,j}^{\ell}$ of M^{3}OFF in different MEC Envs, where the weights for a MEC Env are sampled from multiple episodes. We concatenate $\rho_{i,j}^{\ell}$ from all layers into a $(L-1)\times M^2 = 32$ dimensional vector standing for the composing weights, and then project them into two-dimensional points for visualization using t-SNE as shown in Fig. 11. The DC-output weights of Env2 (green points) are found to be close to Env4 (red points), as well as the weights between Env1 (orange points) and Env3 (pink points). In addition, the weights of Env5 (purple points) are also noticed to be close to Env2 (green points), since they both involve v_2^{env} and v_{10}^{env}. This further demonstrates that M^{3}OFF could distinguish different MEC Envs and share modules for similar Envs.

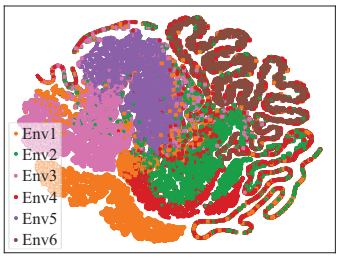
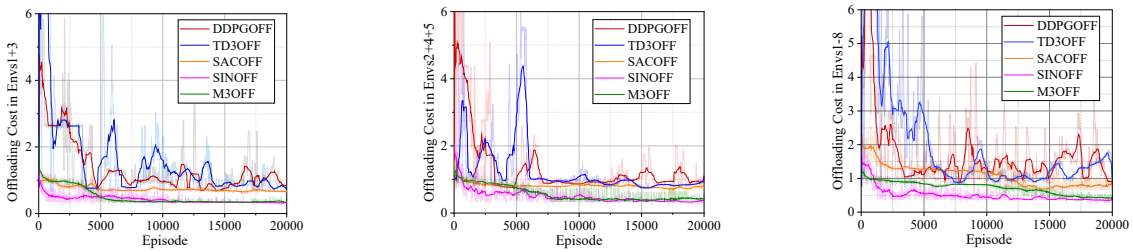


Fig. 11. Visualization of DC-output weights via t-SNE.

C. Performance Comparison with Baselines

We compare M^{3}OFF with 4 baselines in Fig. 12, where DDPGOFF, TD3OFF, SACOFF directly adopt state-of-the-art DRL offloading approaches in three different sizes of Envs while SINOFF individually adopts SAC in each one of Envs and takes the average cost. When $|\mathscr{V}|$ increases from 2 (Fig. 12(a)) to 8 (Fig. 12(c)), DDPGOFF and TD3OFF are both observed to meet converging difficulties in multiple MEC Envs possibly due to the experience-learning interference among multiple Envs, and SACOFF is found to become unstable with high offloading cost around 0.68 when $|\mathscr{V}| = 8$ in Fig. 12(c).

Authorized licensed use limited to: Zhejiang University of Technology. Downloaded on December 13,2024 at 08:58:10 UTC from IEEE Xplore. Restrictions apply.



(a) Performance in Envs1+3. (b) Performance in Envs2+4+5. (c) Performance in Envs1-8.

Fig. 12. Performance comparison between M^{3}OFF and baselines in MEC Envs with different sizes ($|\mathscr{V}| = 2, 5, 8$).

Comparatively, M^{3}OFF is noticed to achieve more promising offloading performance in any of the three types of Envs, and see its offloading cost touch below 0.5 (26% lower than SACOFF) when $|\mathscr{V}| = 8$. In addition, SINOFF is observed to achieve the lowest cost among all compared approaches, primarily because of its individual training and offloading in each of MEC Envs. However, M^{3}OFF is still found to achieve offloading performance close to SINOFF, largely due to the capability of M^{3}OFF to improve learning efficiency by reusing the same modules in similar MEC Envs and reduce learning interference by composing different modules.

D. Real Testbed Experiments

We further evaluate M^{3}OFF on a real testbed, built using a Wi-Fi router (802.11n wireless LAN) as the BS, a workstation powered by 1× Intel Core i9-11900K as the ES, and 1× Raspberry-Pi-4B and 2× robotic vehicles (1× Raspberry-Pi-4B/vehicle) as UEs, as shown in Fig. 13. Face recognition, with three different picture sizes (10/30/60+ KB), is taken as the UE's mobile applications. Three types of MEC Envs are configured for testbed evaluation, including TB-Env1 (10+ KB, 2 MB/s, 2.6 GHz), TB-Env2 (30+ KB, 3 MB/s, 3 GHz) and TB-Env3 (60+ KB, 4 MB/s, 3.4 GHz) where the involved three elements of each TB-Env denote the picture size, transmission rate and workstation frequency, respectively. Considering the small dimension of s_n and a_n in the testbed with small number of UEs compared to numerical simulation, we adopt a 3×4×Module64 M^{3}OFF to reduce modules' capacity and improve learning speeds. We run M^{3}OFF, as well as DDPGOFF, TD3OFF, SACOFF and SINOFF, in TB-Env1, TB-Env1+3, TB-Env2+3 and TB-Env1+2+3, respectively, to evaluate the offloading performance in various MEC Envs.

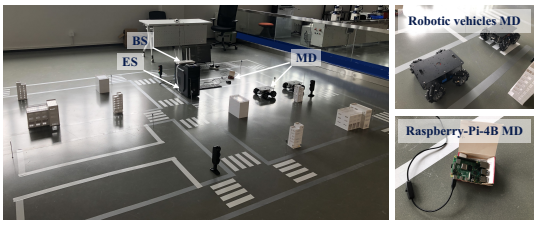
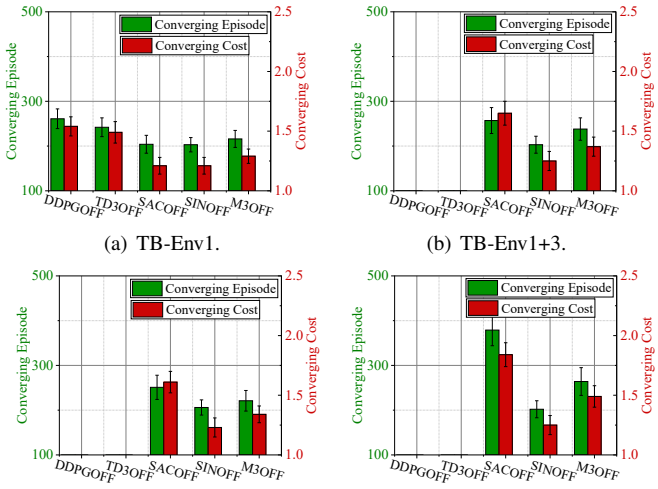


Fig. 13. A real testbed to evaluate multi-environmental MEC offloading.

For the single MEC TB-Env1 in Fig. 14(a), each offloading approach is observed to converge to a cost, where SACOFF and SINOFF are found to achieve the lowest cost around 2.31 at about the 800-th episode while M^{3}OFF achieves a higher cost around 2.45 at about the 1100-th episode. However, when it comes to multiple MEC, e.g., TB-Env1+3, TB-Env2+3 and TB-Env1+2+3 in Figs. 14(b) to 14(d), DDPGOFF and TD3OFF could not see desirable converging costs within limited episodes, and M^{3}OFF is noticed to achieve lower costs than SACOFF by 16.9%, 16.7% and 19.1%, respectively. M^{3}OFF is also found to obtain the cost closest to SINOFF, the individually-trained offloading approach in each TB-Env.



(a) TB-Env1. (b) TB-Env1+3.

(c) TB-Env2+3. (d) TB-Env1+2+3.

Fig. 14. Performance evaluation on the real testbed.

V. Conclusion

We presented a DRL-based multi-environmental computation offloading approach (M^{3}OFF) for realistic MEC scenarios with variant system bandwidth, edge capacity, task types, etc. Facing the learning inefficiency and interference issue of DRL-based offloading approaches in multiple MEC environments, we propose to generate offloading actions based on module composition, instead of a single DRL network, that improves learning efficiency by reusing the same modules and mitigate learning interference by composing distinct modules. We conduct extensive experiments to evaluate and visualize the benefits resulted from module-compositional offloading decision-making, as well as the superiority of M^{3}OFF against state-of-the-art DRL-based offloading approaches.

Authorized licensed use limited to: Zhejiang University of Technology. Downloaded on December 13,2024 at 08:58:10 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[2] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—a key technology towards 5g," *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.

[3] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE J. Sel. Areas Commun.*, vol. 19, no. 3, pp. 1628–1656, 2017.

[4] L. Lin, X. Liao, H. Jin, and P. Li, "Computation offloading toward edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1584–1607, 2019.

[5] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 587–597, 2018.

[6] 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, 2019.

[7] Y. Geng, Y. Yang, and G. Cao, "Energy-efficient computation offloading for multicore-based mobile devices," in *Proc. of INFOCOM*. IEEE, 2018, pp. 46–54.

[8] L. Yang, H. Zhang, M. Li, J. Guo, and H. Ji, "Mobile edge computing empowered energy efficient task offloading in 5g," *IEEE Trans. Veh. Teclmol.*, vol. 67, no. 7, pp. 6398–6409, 2018.

[9] Q. Luo, S. Hu, C. Li, G. Li, and W. Shi, "Resource scheduling in edge computing: A survey," *IEEE J. Sel. Areas Commun.*, vol. 23, no. 4, pp. 2131–2165, 2021.

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

[11] 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. Mobile Comput.*, vol. 19, no. 11, pp. 2581–2593, 2020.

[12] S. Bi, L. Huang, H. Wang, and Y.-J. A. Zhang, "Lyapunov-guided deep reinforcement learning for stable online computation offloading in mobile-edge computing networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 11, pp. 7519–7537, 2021.

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

[14] X. Huang, S. Leng, S. Maharjan, and Y. Zhang, "Multi-agent deep reinforcement learning for computation offloading and interference coordination in small cell networks," *IEEE Trans. Veh. Technol.*, vol. 70, no. 9, pp. 9282–9293, 2021.

[15] N. Waqar, S. A. Hassan, A. Mahmood, K. Dev, D.-T. Do, and M. Gidlund, "Computation offloading and resource allocation in mec-enabled integrated aerial-terrestrial vehicular networks: A reinforcement learning approach," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 11, pp. 21 478–21 491, 2022.

[16] N. Cheng, F. Lyu, W. Quan, C. Zhou, H. He, W. Shi, and X. Shen, "Space/aerial-assisted computing offloading for iot applications: A learning-based approach," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 5, pp. 1117–1129, 2019.

[17] Z. Qin, H. Yao, T. Mai, D. Wu, N. Zhang, and S. Guo, "Multi-agent reinforcement learning aided computation offloading in aerial computing for the internet-of-things," *IEEE Trans. Serv. Comput.*, 2022.

[18] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE J. Sel. Areas Commun.*, vol. 21, no. 4, pp. 3133–3174, 2019.

[19] A. Feriani and E. Hossain, "Single and multi-agent deep reinforcement learning for ai-enabled wireless networks: A tutorial," *IEEE J. Sel. Areas Commun.*, vol. 23, no. 2, pp. 1226–1252, 2021.

[20] L. Li, Q. Cheng, X. Tang, T. Bai, W. Chen, Z. Ding, and Z. Han, "Resource allocation for noma-mec systems in ultra-dense networks: A learning aided mean-field game approach," *IEEE Trans. Wireless Commun.*, vol. 20, no. 3, pp. 1487–1500, 2020.

[21] X. Pang, Z. Wang, J. Li, R. Zhou, J. Ren, and Z. Li, "Towards online privacy-preserving computation offloading in mobile edge computing," in *Proc. of INFOCOM*. IEEE, 2022, pp. 1179–1188.

[22] J. Shi, J. Du, J. Wang, J. Wang, and J. Yuan, "Priority-aware task offloading in vehicular fog computing based on deep reinforcement learning," *IEEE Trans. Veh. Technol.*, vol. 69, no. 12, pp. 16 067–16 081, 2020.

[23] J. S. ROJAS. IP Network Traffic Flows Labeled with 87 Apps. (2023, April 16). [Online]. Available: https://www.kaggle.com/datasets/jsrojas/ip-network-traffic-flows-labeled-with-87-apps

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

[25] J. Andreas, D. Klein, and S. Levine, "Modular multitask reinforcement learning with policy sketches," in *International conference on machine learning*. PMLR, 2017, pp. 166–175.

[26] C. Devin, A. Gupta, T. Darrell, P. Abbeel, and S. Levine, "Learning modular neural network policies for multi-task and multi-robot transfer," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 2169–2176.

[27] R. Yang, H. Xu, Y. Wu, and X. Wang, "Multi-task reinforcement learning with soft modularization," *Advances in Neural Information Processing Systems*, vol. 33, pp. 4767–4777, 2020.

[28] L. Shen, "User experience oriented task computation for uav-assisted mec system," in *Proc. of INFOCOM*. IEEE, 2022, pp. 1549–1558.

[29] Y. S. Nasir and D. Guo, "Multi-agent deep reinforcement learning for dynamic power allocation in wireless networks," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 10, pp. 2239–2250, 2019.

[30] F. Meng, P. Chen, L. Wu, and J. Cheng, "Power allocation in multi-user cellular networks: Deep reinforcement learning approaches," *IEEE Trans. Wireless Commun.*, vol. 19, no. 10, pp. 6255–6267, 2020.

[31] P. Dent, G. E. Bottomley, and T. Croft, "Jakes fading model revisited," *Electronics letters*, vol. 13, no. 29, pp. 1162–1163, 1993.

[32] S. Jošilo and G. Dán, "Computation offloading scheduling for periodic tasks in mobile edge computing," *IEEE/ACM Trans. Netw.*, vol. 28, no. 2, pp. 667–680, 2020.

[33] D. Van Huynh, V.-D. Nguyen, S. R. Khosravirad, V. Sharma, O. A. Dobre, H. Shin, and T. Q. Duong, "Urllc edge networks with joint optimal user association, task offloading and resource allocation: A digital twin approach," *IEEE Trans. Commun.*, vol. 70, no. 11, pp. 7669–7682, 2022.

[34] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Trans. Commun.*, vol. 64, no. 10, pp. 4268–4282, 2016.

[35] Y. Chen, J. Zhao, Y. Wu, J. Huang, and X. S. Shen, "Qoe-aware decentralized task offloading and resource allocation for end-edge-cloud systems: A game-theoretical approach," *IEEE Trans. Mobile Comput.*, 2022.

[36] W. Chu, P. Yu, Z. Yu, J. C. Lui, and Y. Lin, "Online optimal service selection, resource allocation and task offloading for multi-access edge computing: A utility-based approach," *IEEE Trans. Mobile Comput.*, 2022.

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

[38] N. Zhao, Z. Ye, Y. Pei, Y.-C. Liang, and D. Niyato, "Multi-agent deep reinforcement learning for task offloading in uav-assisted mobile edge computing," *IEEE Trans. Wireless Commun.*, vol. 21, no. 9, pp. 6949–6960, 2022.

[39] X. He, S. Wang, X. Wang, S. Xu, and J. Ren, "Age-based scheduling for monitoring and control applications in mobile edge computing systems," in *Proc. of INFOCOM*. IEEE, 2022, pp. 1009–1018.

[40] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. of ICML*. PMLR, 2018, pp. 1861–1870.

[41] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

Authorized licensed use limited to: Zhejiang University of Technology. Downloaded on December 13,2024 at 08:58:10 UTC from IEEE Xplore. Restrictions apply.