FEAT: Towards Fast Environment-Adaptive Task Offloading and Power Allocation in MEC

Tao Ren*, Zheyuan Hu†, Hang He†, Jianwei Niu†‡§, Xuefeng Liu†‡
* Laboratory for Internet Software Technologies, 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, hehang, niujianwei, liu_xuefeng}@buaa.edu.cn

IEEE INFOCOM 2023 - IEEE Conference on Computer Communications | 979-8-3503-3414-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/INFOCOM53939.2023.10228946

***Abstract*—Mobile edge computing (MEC) has been proposed to provide mobile devices with both satisfactory computing resources and latency. Key issues in MEC include task offloading and power allocation (TOPA), for which deep reinforcement learning (DRL) is becoming a popular methodology. However, most DRL-based TOPA approaches are typically developed in a certain environment, lacking the adaptability to unseen environments. Motivated by this, this paper proposes a Fast Environment-Adaptive TOPA (FEAT) approach that could adapt to unseen environments with little fine-tuning. Specifically, we first split MEC states into the internal state and environmental state. Then, based on these two types of states, we develop two main components of FEAT — a group of internal state-dependent TOPA meta-policies and an environmental state-embedded steerer. Meta-policies learn TOPA skills within the internal state space (allowing reusing meta-policies in different environments), while the steerer learns to choose appropriate meta-policies according to embedded environmental states. When encountering an unseen environment with the same internal state space, FEAT only needs to fine-tune the steerer using the newly embedded environmental state with few internal state explorations. Extensive experimental results on simulation and testbed demonstrate that FEAT outperforms the state-of-the-art by more than 16.4% in terms of fine-tuning speeds.**

***Index Terms*—Mobile Edge Computing, Task Offloading, Power Allocation, Environment Adaptation**

I. Introduction

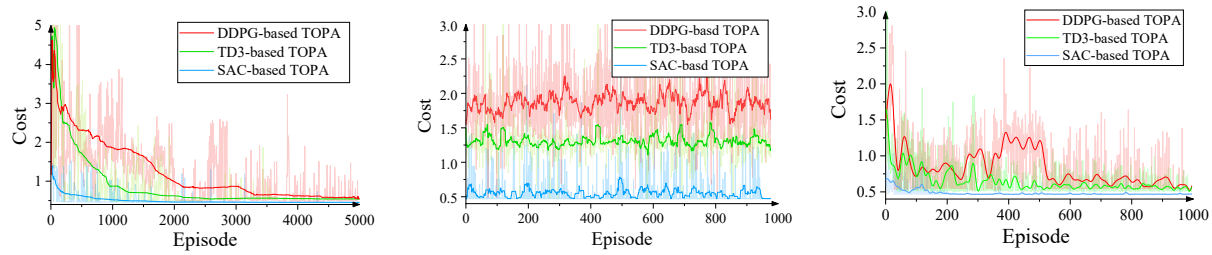
Recent years have witnessed the booming development of mobile devices (MDs), as well as the popularity of mobile applications, e.g., face recognition, mobile games [1]. However, there is an increasing gap between MD's capability and mobile application's requirements, giving rise to the proposal of mobile edge computing (MEC) as a promising paradigm to provide MDs with both satisfactory computing resources and latency [2]. Key issues in MEC include task offloading (which tasks in MDs should be offloaded to the edge server) and power allocation (how much transmitting power should be adopted), so that MDs could achieve satisfactory task execution with as little energy consumption as possible [3].

This work was supported in part by Beijing Municipal Science and Technology Program under Grant No. Z221100007722001 and National Natural Science Foundation of China under Grant No. 61976012. (Corresponding author: Jianwei Niu.)

Great efforts have been made by researchers on studying task offloading and power allocation (TOPA) [4], [5], which is solved usually using exhaustive search methods [6]–[8] or mathematical optimization techniques [9], [10]. Nonetheless, these two types of approaches rely on either high computational complexity or dependable mathematical models, thus are hard to adapt to high-speed uncertain MEC networks. Facing the challenge, more and more researchers resort to developing TOPA based on deep reinforcement learning (DRL) [11], [12], a combination of deep learning and reinforcement learning suitable for solving high-dimension Markov decision process (MDP) under uncertain stochastic scenarios. There has been a wide variety of DRL-based TOPA studies, from binary offloading [13] to partial offloading [14], single base station (BS) [15] to multiple BSs [16], and edge server (ES) assisted [17] to end-edge-cloud orchestrated [18], etc.

However, most existing DRL-based TOPA approaches are developed in a certain MEC environment with fixed bandwidth, ES computing capacity or task arrival pattern, where only task sizes and channel conditions could be time-varying. When applied in different MEC environments, these approaches would suffer from severe performance degradation. As shown in Fig. 1, a preliminary experiment is conducted with respect to TOPA based on three popular DRL algorithms [19]–[21], i.e., DDPG, TD3 and SAC, respectively. In Fig. 1(a), three approaches are trained in an environment with 10 MHz bandwidth, 9 GHz ES capacity, $\mathcal{N}(700, 1000)$ Kb task arrival pattern, and seen to converge in the episode 3300, 2500, 1200 and to the cost (weighted sum of task delay and energy consumption, with the same weight) 0.58, 0.54, 0.46, respectively. In Fig. 1(b), three trained approaches are directly applied in a different MEC environment with 9 MHz bandwidth, 8.5 GHz ES capacity, $\mathcal{N}(700, 6000)$ Kb task arrival pattern, resulting in a noticeable cost rise to 1.8, 1.3, 0.6 (about 210%, 140%, 30% increase), respectively. In Fig. 1(c), three trained approaches from Fig. 1(a) are fine-tuned in the new environment of Fig. 1(b) and observed to re-converge to the cost 0.64, 0.58, 0.49 (approximately 10.3%, 7.4%, 6.5% increase) in the episode 550, 310, 200, respectively. This type of performance degradation and long-time fine-tuning of DRL-based TOPA approaches in new environments significantly weaken their practicality in realistic scenarios, since it is unrealistic to train

Authorized licensed use limited to: Zhejiang University of Technology. Downloaded on September 23,2024 at 12:03:53 UTC from IEEE Xplore. Restrictions apply.



(a) Training in a certain MEC environment. (b) Running directly in a different environment. (c) Fine-tuning in a different environment.

Fig. 1. A preliminary experiment investigating the adaptability of three popular DRL-based TOPA approaches during training, directly running, and fine-tuning.

a new DRL approach or fine-tune a previous approach for a long time for each distinct MEC environment. To the best of our knowledge, there has been scarce research focusing on the study of environment-adaptive DRL-based TOPA approaches.

Motivated by this, this paper proposes a Fast Environment-Adaptive TOPA (FEAT) approach, that could adapt to unseen MEC environments with little fine-tuning. Specifically, we first distinguish the internal states and environmental states in MEC, which are then utilized to facilitate the design of two main components of FEAT — a group of internal state-dependent TOPA meta-policies and an environmental state-embedded steerer. Then, through learning from TOPA experiences in a diverse set of MEC environments, each meta-policy learns to make desirable TOPA decisions on a cluster of internal states, the whole of which is covered by the ensemble of meta-policies, and the steerer learns to choose an appropriate meta-policy according to the embedded environmental states. Last, when encountering an unseen environment with the same internal state space, FEAT just needs to fine-tune the steerer using the new embedded environmental state with few internal state explorations. Extensive experiments are conducted on both simulation and testbed, and the results demonstrate that FEAT could outperform state-of-the-art DRL-based TOPA approaches by more than 16.4%, w.r.t. fine-tuning speeds in unseen MEC environments. The main contributions of this paper are summarized as follows:

- We distinguish distinct types of states in MEC, including the time-varying internal state, time-varying external state and time-invariant external state, the last two of which support the definition of the environmental state, to facilitate the problem formulation and transformation of environment-adaptive DRL-based task offloading and power allocation.
- We design a fast environment-adaptive TOPA (FEAT) approach based on ensemble DRL and environment embedding, consisting of a high-level steerer responsible for choosing an appropriate meta-policy based on its observation of the environmental state and a group of meta-policies skillful at making TOPA actions in distinct clusters of internal states. Moreover, we make an elaborate design on the training algorithm of FEAT to enhance its converging performances.
- We conduct extensive experiments to evaluate the training components of FEAT and make a deep look into the roles of distinct TOPA meta-policies through state and action visualization, after which the adaptability of FEAT against the state-of-the-art DRL-based TOPA is verified in various unseen environments. Furthermore, we demonstrate the performance advantages of FEAT over the state-of-the-art on a real testbed.

II. System Model and Problem Formulation

In this section, we present a typical MEC system to support the investigation of fast adaptive task offloading and power allocation. As shown in Fig. 2, there are a BS connected with an ES with computing capacity f^{ES} via a high-speed wired link, and M MDs. We denote the set of MDs as $\mathscr{M} = \{\text{MD}^1, \text{MD}^2, ..., \text{MD}^M\}$, where MD^m represents the m-th MD. The finite time horizon T in one episode is discretized into N equal time slots with length $\delta = T/N$, the set of which is denoted by $\mathbb{N} = \{1, 2, ..., N\}$. In each slot $i \in \mathbb{N}$, a task $\Gamma_i^m = \{d_i^m, c, t^{\max}\}$ arrives at MD^m, where d_i^m is the task size (in bits), c and $t^{\max}$ are the amount of CPU cycles per bit of task data and maximum tolerable latency of the task, respectively, and assumed to be the same for all tasks. In slot i, each MD^m can choose to either execute Γ_i^m locally or offload (with transmitting power p_i^m below the maximum p^{MD}) it to BS for edge computing through wireless transmission with system bandwidth B, and the decision is denoted by an offloading variable $a_i^m \in \{0, 1\}$ with 0 representing local execution and 1 edge computing. The computing capacity and initial energy budget of each MD is denoted as f^{MD} and eb^{MD}, respectively, and the energy budget of MD^m is time-varying due to its energy consumption and denoted as eb_i^m in slot i. Besides, each MD^m is equipped with a task queue whose length q_i^m stands for the amount of task size waiting for execution, and so is it for ES with a queue length q_i^{ES}.

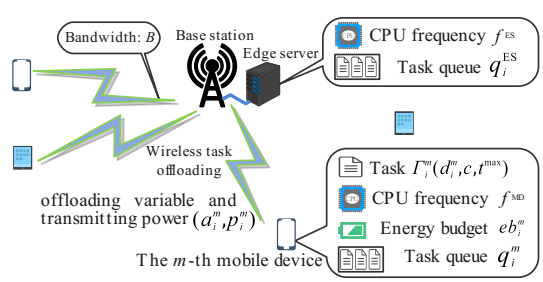


Fig. 2. An example of a MEC network.

A. MEC System Model

1) Local Computing Model

If MD^m chooses to execute Γ_i^m locally, i.e., $a_i^m = 0$, the local computation delay $t_i^{m,\text{loc}}$ of Γ_i^m, including the waiting time and executing time, can be calculated as

$$t_i^{m,\text{loc}} = \frac{(q_i^m + d_i^m)c}{f^{\text{MD}}}, \tag{1}$$

Authorized licensed use limited to: Zhejiang University of Technology. Downloaded on September 23,2024 at 12:03:53 UTC from IEEE Xplore. Restrictions apply.

and the corresponding energy consumption can be given by

$$e_i^{m,\text{loc}} = \xi (f^{\text{MD}})^2 d_i^m c, \tag{2}$$

where ξ is the energy efficiency coefficient of MD's chip [22].

2) Edge Computing Model

If MD^m chooses to offload Γ_i^m to BS, i.e., $a_i^m = 1$, its edge computation delay $t_i^{m,\text{edge}}$ should include three parts: the wireless transmitting time $t_i^{m,\text{tra}}$, the waiting time $t_i^{m,\text{eswait}}$ in ES and the executing time $t_i^{m,\text{esexe}}$ in ES.

First, concerning wireless transmission, the channel gain g_i^m between the BS and MD^m in slot i can be given by $g_i^m = \beta_i^m |h_i^m|^2$, where β_i^m is the large-scale fading component, and h_i^m represents the small-scale Rayleigh fading component statistically modeled as a Gaussian process based on the Jakes model, details of which could be found in [23]. Following 3GPP LTE specifications, β_i^m is modeled as $\beta_i^m = -148.1 - 37.6 \log_{10} \ell_i^m + z$, where ℓ_i^m is the distance (measured in km) between the BS and MD^m and z is a log-normal random variable obeying $\mathcal{N}(0, \sigma_z^2)$ [24]. With g_i^m, the signal-to-interference-plus-noise ratio (SINR) is calculated as

$$\text{SINR}_i^m = \frac{g_i^m p_i^m}{\sum_{m' \neq m} g_i^{m'} p_i^{m'} + \sigma_{\text{SINR}}^2}, \tag{3}$$

where σ_{SINR}^2 is the additive noise power. By now, the transmitting time $t_i^{m,\text{tra}}$ for offloading Γ_i^m to BS can be given by

$$t_i^{m,\text{tra}} = \frac{d_i^m}{B \log(1 + \text{SINR}_i^m)}. \tag{4}$$

Second, the waiting time $t_i^{m,\text{eswait}}$ of the offloaded Γ_i^m in ES depends on two parts: One is the current queue size q_i^{ES}, and the other is all sizes $q_i^{\text{bef},m}$ of offloaded tasks completing their transmission before Γ_i^m, i.e., $q_i^{\text{bef},m} = \sum_{m' \neq m} \mathbb{1}(t_i^{m',\text{tra}} < t_i^{m,\text{tra}}) d_i^{m'}$. Therefore, $t_i^{m,\text{eswait}}$ could be represented by

$$t_i^{m,\text{eswait}} = \frac{(q_i^{\text{ES}} + q_i^{\text{bef},m}) c}{f^{\text{ES}}}. \tag{5}$$

Third, the executing time $t_i^{m,\text{esexe}}$ of the offloaded Γ_i^m in ES can be directly given by

$$t_i^{m,\text{esexe}} = \frac{d_i^m c}{f^{\text{ES}}}. \tag{6}$$

From Eqs. (4) to (6), we get the edge computation delay

$$t_i^{m,\text{edge}} = t_i^{m,\text{tra}} + t_i^{m,\text{eswait}} + t_i^{m,\text{esexe}}. \tag{7}$$

Here, similar to most existing work [9], [25], the downloading time of task results is not considered since the result size is generally much smaller than d_i^m, and so is the energy consumption of MD^m during its waiting for edge-computing Γ_i^m due to the comparatively low power in idle state. Hence, the energy consumption of Γ_i^m in edge computing, which only takes MD^m into account due to the relatively huge energy supply of ES, can be given by

$$e_i^{m,\text{edge}} = p_i^m t_i^{m,\text{tra}}. \tag{8}$$

3) System Dynamics and Metrics

In each slot i, the task queue q_i^m and energy budget eb_i^m of each MD^m, along with the ES task queue q_i^{ES} will change dynamically according to the variation of a_i^m and p_i^m (if selected as TOPA variables) as follows:

$$\begin{cases} eb_{i+1}^m = eb_i^m - \left[(1 - a_i^m) e_i^{m,\text{loc}} + a_i^m e_i^{m,\text{edge}}\right] \\ q_{i+1}^m \ = \max\{0,\ q_i^m + d_i^m - \dfrac{f^{\text{MD}} \delta}{c}\} \\ q_{i+1}^{\text{ES}} \ = \max\{0,\ q_i^{\text{ES}} + \sum\nolimits_{m=1}^{M} a_i^m d_i^m - \dfrac{f^{\text{ES}} \delta}{c}\} \end{cases}. \tag{9}$$

In addition, the system metrics in each slot primarily lie in two terms, i.e., t_i (average delay) and e_i (average energy consumption), which are formulated as

$$\begin{cases} t_i = \dfrac{1}{M} \displaystyle\sum_{m=1}^{M} \left[(1 - a_i^m) t_i^{m,\text{loc}} + a_i^m t_i^{m,\text{edge}}\right] \\ e_i = \dfrac{1}{M} \displaystyle\sum_{m=1}^{M} \left[(1 - a_i^m) e_i^{m,\text{loc}} + a_i^m e_i^{m,\text{edge}}\right] \end{cases}. \tag{10}$$

B. Internal State and Environmental State

Given the above MEC model, system dynamics are driven by Eq. (9), during the operation of which a TOPA action (a_i^m, p_i^m) is chosen according to the observed MEC state (e.g., d_i^m, eb_i^m, q_i^m, q_i^{ES}, B, f^{MD}, f^{ES}), with the goal of some objective(s). In turn, some variables (eb_i^m, q_i^m and q_i^{ES}) of the state, but not all, will be affected by the chosen action (a_i^m, p_i^m) and change their values accordingly, driven by the system dynamics. Hence, we term these affected variables *time-varying internal state* $s_i^{\text{in}} = (eb_i^m, q_i^m, q_i^{\text{ES}})$, while other unaffected elements *time-varying external state* $s_i^{\text{ext}} = d_i^m$ and *time-invariant external state* $s_i^{\text{inv}} = (B, f^{\text{MD}}, f^{\text{ES}})$. That is to say, the external states, whether time-varying s_i^{ext} or time-invariant s_i^{inv}, change without the control of system dynamics, but according to the environment setting. Without confusion, we uniformly refer to the external states as environmental state Env, i.e., $Env = (d_i^m, B, f^{\text{MD}}, f^{\text{ES}})$.

Within Env, the time-varying external state s_i^{ext}, i.e., d_i^m, is usually defined in most existing work following some distributions, e.g., Gaussian distribution [26] defined by μ and σ^2, uniform distribution [20] defined by d^{min} and d^{max}, or Poisson distribution [27] defined by $p^{\text{arrivrate}}$. For brevity, we suppose $d_i^m \sim D(\Lambda)$, where Λ can be a scalar or vector. Thus, to this end, the MEC environment can be formulated as a *time-invariant environmental state*[1]

$$Env = (\Lambda, B, f^{\text{MD}}, f^{\text{ES}}), \tag{11}$$

meaning that each environment $\varepsilon \in Env$ is a group of definite values of Λ, B, f^{MD}, f^{ES}, and can differ from one another. Considering the focus of this paper, we are dedicated to develop an adaptive TOPA approach that can achieve desirable performances across different $\varepsilon \in Env$.

[1]If $c, t^{\text{max}} \in \Gamma$ are also time-varying and follow some distributions, Env can be easily extended by cascading the distribution parameters.

Authorized licensed use limited to: Zhejiang University of Technology. Downloaded on September 23,2024 at 12:03:53 UTC from IEEE Xplore. Restrictions apply.

C. Problem Formulation and DRL-based TOPA

1) Problem Formulation: Since the focus of this paper is to design adaptive TOPA approaches rather than investigate specific TOPA objectives, we assume a typical objective, namely the minimization of the cost C_i defined as the weighted sum of task latency and energy consumption, i.e.,

$$C_i = \eta t_i + (1-\eta) e_i. \tag{12}$$

As a result, the optimization problem $\mathscr{P}$ is formulated as:

$$\begin{aligned} \mathscr{P}: \min_{a_i^m, p_i^m} \quad & \sum_{i=1}^{N} C_i(a_i^m, p_i^m) && (13)\\ \text{s.t.} \quad & C1: \ a_i^m \in \{0,1\}, && (14)\\ & C2: \ 0 \le p_i^m \le p^{\text{MD}}, && (15)\\ & C3: \ (1-a_i^m) t_i^{m,\text{loc}} + a_i^m t_i^{m,\text{edge}} \le t^{\max}, && (16)\\ & C4: \ eb_i^m \ge 0, && (17) \end{aligned}$$

where the constraint $C1$ describes the binary offloading mode, $C2$ manifests that the MD's transmitting power should be less than the maximum available power, $C3$ ensures each task's computation delay must be less than the maximum tolerable latency, and $C4$ guarantees each MD's available energy budget in any slot should be no less than zero.

$\mathscr{P}$ can be seemed as a knapsack problem: There are totally $\sum_{m=1}^{M}\sum_{i=1}^{N} d_i^m$ weights of items, each item with weight d_i^m needs to be placed in MD^m or ES with different costs, and how to place all the items to minimize the summed costs without violating any constraint (e.g., the task latency, energy consumption). Multiplied with the assignment of p_i^m, the problem could be much harder.

2) Markov-Oriented Problem Transformation: $\mathscr{P}$ could be solved from the perspective of MDP, which is denoted by a tuple (S, A, P, r) with each element standing for the state space, action space, system dynamics and immediate reward, respectively. The objective of MDP is to maximize the accumulated rewards $\sum_{i=1}^{N} r_i$, making it possible to transform $\mathscr{P}$ into an MDP problem $\mathscr{P}1$ since minimizing $\sum_{i=1}^{N} C_i$ is equal to maximizing $\sum_{i=1}^{N} r_i$ letting $r_i = C_i^{-1}$. In practice, r_i is usually discounted using a factor γ, indicating the importance of the future reward. Therefore, the transformed problem $\mathscr{P}1$ can be represented as

$$\mathscr{P}1: \max_{a_i^m, p_i^m} \sum_{i=1}^{N} \gamma^{i-1} r_i. \tag{18}$$

3) MDP-based TOPA Scheme: To develop DRL-based TOPA scheme, S, A and r are designed as follows.

State: In each slot, DRL makes TOPA decisions according to its observation, consisting of the observed time-varying variable s_i^{obs} that includes the time-varying internal state s_i^{in} and time-varying external state d_i^m, i.e., $s_i^{\text{obs}} = (s_i^{\text{in}}, d_i^m)$, along with the time-invariant environmental state Env. Hence, the DRL state in slot i is defined as $s_i = (s_i^{\text{obs}}, Env) = (s_i^{\text{in}}, d_i^m, Env) = (eb_i^m, q_i^m, q_i^{\text{ES}}, d_i^m, \Lambda, B, f^{\text{MD}}, f^{\text{ES}})$, where $m \in \mathscr{M}$.

Action: DRL adjusts each MD's offloading variable and transmitting power, aiming at maximizing the accumulated rewards. Thus, we define the action in slot i as $a_i = (a_i^m, p_i^m)$, where $m \in \mathscr{M}$.

Reward: After taking a_i in s_i, a reward inversely proportional to C_i, is generated if Eqs. (14) to (17) are all satisfied; Otherwise, a penalty corresponding to the violated constraint, will be generated. Thus, we define $r_i = \mathbb{1}(\cdot)\alpha(C_i^{-1} - C_{i,\text{local}}^{-1})/C_{i,\text{local}}^{-1} + (1-\mathbb{1}(\cdot))\sum_{j=1}^{4}\beta^j r_j^{\text{penal}}$, where α is the coefficient of the (normalized) immediate reward, $C_{i,\text{local}}$ is the cost for local computing all Γ_i^m, β^j is the weight of each constraint, r_j^{penal} is a penalty proportional to the degree to which Cj has been violated, and $\mathbb{1}(\cdot)$ is an indicator function equaling 1 if all constraints are satisfied and 0 otherwise.

4) Learning Parameterized Scheduling Model: Let $Q(s_i, a_i)$ denote the expected accumulated rewards after taking a_i in s_i, i.e., $Q(s_i, a_i) = \sum_{j=i}^{j=N} \gamma^{j-1} r_j$, then $Q(s_i, a_i)$ can be written in an iterative way according to the Bellman equation:

$$Q(s_i, a_i) \triangleq \mathbb{E}_{s_{i+1}\sim P^{\varepsilon}}\big[r_i + \gamma \max_{a_{i+1}\sim\pi} Q(s_{i+1}, a_{i+1})\big], \tag{19}$$

where P^{ε} stands for the system dynamics (i.e., state transition probability) in the environment $\varepsilon \in Env$, and $\pi(a_i|s_i)$ is a TOPA policy that determines a_i in s_i. Leveraging deep learning, $Q(s_i, a_i)$ and $\pi(a_i|s_i)$ can be both approximated using neural networks with parameters θ and ϕ, leading to the parameterized $Q^{\theta}(s_i, a_i)$ and $\pi^{\phi}(a_i|s_i)$.

Targeted at minimizing the expected costs in a MEC episode, i.e., maximizing $Q^{\theta}(s_i) = \mathbb{E}_{a_i\sim\pi^{\phi}} Q^{\theta}(s_i, a_i)$ where $i = 0$, the desired TOPA model could be learned via

$$\begin{aligned} \pi^{\phi *}(a_i|s_i) &= \arg\max_{\pi^{\phi}} \mathbb{E}_{a_i\sim\pi^{\phi}} Q^{\theta}(s_i, a_i) && (20)\\ &= \arg\max_{\pi^{\phi}} \mathbb{E}_{a_i\sim\pi^{\phi}, s_{i+1}\sim P^{\varepsilon}}\big[r_i + \gamma \max_{a_{i+1}\sim\pi^{\phi}} Q^{\theta}(s_{i+1}, a_{i+1})\big]. \end{aligned}$$

Stochastic gradient descent methods can be employed to update θ and ϕ by sampling a_i and s_i from π^{ϕ} and P^{ε}, respectively, during which target parameters $\bar{\theta}$ and $\bar{\phi}$ could be taken to stabilize the learning process and periodically soft-updated by θ and ϕ like DQN [28] and DDPG [29] do. These give rise to the Q^{θ} loss function definition using Bellman residual as

$$L_{Q^{\theta}} = \mathbb{E}_{a_i\sim\pi^{\phi}, s_{i+1}\sim P^{\varepsilon}} ||Q^{\theta}(s_i, a_i) - (r_i + \gamma \max_{a_{i+1}\sim\pi^{\bar{\phi}}} Q^{\bar{\theta}}(s_{i+1}, a_{i+1}))||_2, \tag{21}$$

and the corresponding stochastic gradient derivation of ϕ as

$$\nabla_{\phi}\pi^{\phi} = \mathbb{E}_{a_i\sim\pi^{\phi}}\big[-Q^{\bar{\theta}}(s_i, a_i)\nabla_{\phi}\log\pi^{\phi}(a_i|s_i)\big]. \tag{22}$$

From Eqs. (21) and (22), it can be seen that s_i could only be sampled from P^{ε} in limited environments $\varepsilon \in Env$. However, the learned TOPA policy $\pi^{\phi *}$ is expected to work well in various unseen environments with slight fine-tuning or even without, *which is also the key objective of FEAT.*

III. Fast Environment-Adaptive TOPA

This section will present FEAT in detail, including the policy design, model architecture and training algorithm, respectively.

Authorized licensed use limited to: Zhejiang University of Technology. Downloaded on September 23,2024 at 12:03:53 UTC from IEEE Xplore. Restrictions apply.

A. Environment-Adaptive TOPA Policy Overview

The irremovable gap between infinite $\varepsilon \in Env$ and limited sampling of ε, makes it difficult to learn an experience-driven DRL model, expressed by Eq. (20), by exploring all $\varepsilon \in Env$. Leveraging the idea of ensemble learning, we attempt to design a TOPA policy, composed of multiple meta-policies that are good at determining a_i on different clusters of latent internal states s_i^{in} and a high-level steerer that is responsible for choosing the appropriate meta-policy based on the observation of ε and s_i^{in}. Let $\pi^{\phi_k}(a_i|s_i^{\text{in}})$ denote the k-th *meta-policy* parameterized by ϕ_k where $k \in \{1, 2, ..., K\}$ and K is the number of meta-policies, and $\omega^{\psi}(k|s_i^{\text{in}}, \varepsilon)$ denote the *steerer* parameterized by ψ satisfying $\sum_{k=1}^{K} \omega^{\psi}(k|s_i^{\text{in}}, \varepsilon) = 1$ for any s_i^{in} and ε, then the ensemble TOPA policy π can be given by

$$\pi^{\psi,\phi}(a_i|s_i^{\text{in}}, \varepsilon) = \sum_{k=1}^{K} \omega^{\psi}(k|s_i^{\text{in}}, \varepsilon)\pi^{\phi_k}(a_i|s_i^{\text{in}}). \tag{23}$$

Considering the time-varying attribute of $s_i^{\text{ext}} = d_i^m$ missed in both the time-varying s_i^{in} and time-invariant ε, we augment the policy in Eq. (23) with s_i^{ext} as follows,

$$\pi^{\psi,\phi}(a_i|s_i^{\text{var}}, \varepsilon) = \sum_{k=1}^{K} \omega^{\psi}(k|s_i^{\text{var}}, \varepsilon)\pi^{\phi_k}(a_i|s_i^{\text{var}}), \tag{24}$$

where $s_i^{\text{var}} = (s_i^{\text{in}}, s_i^{\text{ext}})$ stands for the time-varying state including both internal and external states.

It is noteworthy that we adopt an asymmetric form between the observations of meta-policies π^{ϕ_k} and steerer ω^{ψ}, where ω^{ψ} can access the environmental state ε (i.e., environment embedded) while π^{ϕ_k} only the time-varying state s_i^{var}. The most significant benefit from this asymmetric design is to avoid the degeneration of a single meta-policy taking charge of all environments, instead encourage the meta-policies to learn distinct skills concerning different clusters of s_i^{var} that can then be integrated by ω^{ψ} according to the need of a given ε. This potentially allows the transfer and reuse of a meta-policy in a similar cluster of states from different εs. When encountering an unseen environment ε', the ϕ_k ($k \in \{1, 2, ..., K\}$) are held unchanged for fear of skill loss, and ψ will be tuned in light of the new environment ε' and the corresponding observed s_i^{var}.

B. Model Architecture

With the above, we build the model of FEAT, as shown in Fig. 3, that can produce a suit of meta-policies π^{ϕ_k} ($k \in \{1, 2, ..., K\}$) and a steerer ω^{ψ}, with desirable environment-adaptive TOPA ability. The following aspects, some of which are not explicitly depicted in Fig. 3, should be noted:

1) The actor-critic learning component of each meta-policy (including θ and π) is implemented using the state-of-the-art off-policy DRL algorithm SAC [30] based on the maximum entropy RL framework, where $Q^{\theta}(s_i^{\text{var}}, a_i)$ is replaced by $Q^{\theta}(s_i^{\text{var}}, a_i) + \zeta\mathscr{H}$ with $\mathscr{H} = -\log \pi^{\phi}(a_i|s_i^{\text{var}})$ whose importance is controlled by a temperature factor ζ.

2) The output of each meta-policy's actor π^k is a pair of vectors (μ^k, σ^k), with each vector μ^k (σ^k) denoting the mean (std) of a set of normal distributions and the dimension $\dim(\mu^k)$ or $\dim(\sigma^k)$ of each vector equaling to the dimension of the action $\dim(a_i)$. Hence, each action a_i is sampled from a multi-modal normal distribution $\mathcal{N}(\mu^k, \sigma^k)$ during exploration.

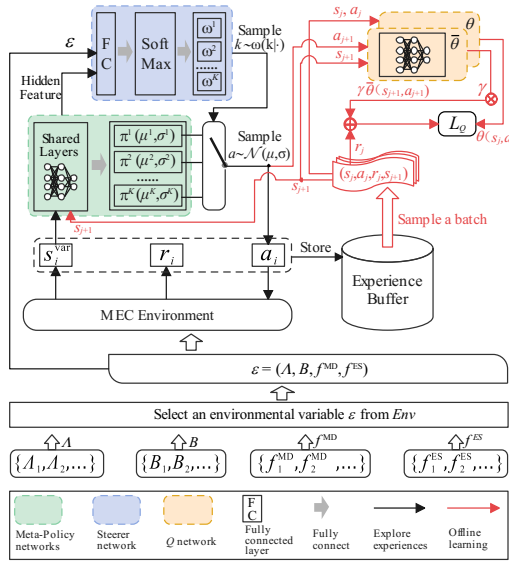


Fig. 3. The model architecture of FEAT.

3) For the π^{ϕ_k} network, hidden layers are shared among meta-policies, distinctions between which appear in the last layer. Besides, instead of directly inputting s_i^{var} along with the embedded $\varepsilon \in Env$ to the ω^{ψ} network, we substitute s_i^{var} with the hidden features extracted from s_i^{var} by the π^{ϕ} network.

4) The dual networks of Q^{θ} and $Q^{\bar{\theta}}$ are not depicted, as well as the target network of π^{ϕ}, for clarity. Additionally, the experience $(s_i^{\text{var}}, a_i, r_i, s_{i+1}^{\text{var}})$ stored into the experience buffer in each slot omits the element s_{i+1}^{var} for brevity, since s_{i+1}^{var} appears at the same position as s_i^{var} in the following slot.

C. Training Algorithm

To train FEAT, there still exists the following intractable issues that need to be addressed during the offline learning of ψ, ϕ and θ networks based on a batch of experience[2] $\mathscr{e} = (s_j, a_j, r_j, s_{j+1})$ sampled from the experience buffer $\mathscr{B}$:

1) ***Stochastic Gradient Estimation for Steerer:*** During the forward experience exploration, the k-th meta-policy is chosen by sampling on the categorical probability distribution of $\omega^{\psi}(k|s_i, \varepsilon)$, which is non-differentiable due to the discontinuity between meta-policies and poses a significant challenge for backward gradient computation [31]. Therefore, we adopt Gumbel-Softmax [32], a popular stochastic gradient estimator, to approximate $\omega^{\psi}(k|s_i, \varepsilon)$ as follows

$$z^{\psi}(k|\cdot) \simeq \frac{\exp((g_i + \log \omega^{\psi}(k|\cdot))/\tau)}{\sum_{k=1}^{K} \exp((g_i + \log \omega^{\psi}(k|\cdot))/\tau)},$$

where $g_i = -\log(-\log(u))$ ($k \in \{1, 2, ..., K\}$) with u sampled uniformly from (0,1), and τ is a softmax temperature annealed via a fixed schedule.

[2]We will omit the superscript 'var' in s_j^{var} and s_{j+1}^{var} hereinafter for brevity.

Authorized licensed use limited to: Zhejiang University of Technology. Downloaded on September 23,2024 at 12:03:53 UTC from IEEE Xplore. Restrictions apply.

In this way, ω^{ψ} could be optimized with the loss

$$L_{z^{\psi}}=\mathbb{E}_{s_j,a_j\sim\mathscr{B}}\Big[\frac{1}{K}\sum_{k=1}^{K}\big(\mathbb{1}(k=\arg\max_{\kappa}Q^{\bar{\theta}_{\kappa}}(s_j,a_j))-z(k|s_j,\varepsilon)\big)^2\Big], \quad (25)$$

where ε is the environmental state[3] in which e was explored and generated. Guided by $L_{z^{\psi}}$, the parameter ψ of the steer is optimized at the direction of choosing the meta-policy that can maximize the accumulated rewards with given s_j, ε and a_j. An alternative form of $L_{z^{\psi}}$ (not presented to save space) could be that $\mathbb{1}(\cdot)$ is substituted with the corresponding k[4] (that generated a_j) assigned by 1 and others by 0. This alternative would encourage ψ to be optimized at the direction that could fit the historical decision of z^{ψ}. We denote this two kinds of loss as L_{ψ}^{CMQ} and L_{ψ}^{HSD}, respectively, and will empirically investigate their performances in the experiments.

*2) **Meta-Policy Selection for Gradient Back-Propagation**:* From Eq. (22), it can be seen that, given a batch of e, $-Q^{\bar{\theta}}(j_j,a_j)$ is deterministic while $\nabla_{\phi}\pi^{\phi}(a_j|s_j)$ needs a selection of ϕ_k from $k\in\{1,2,...,K\}$, i.e., from K meta-policy networks. There are three possible selections:

(1) The first and second, similar to the selection of L_{ψ}^{CMQ} and L_{ψ}^{HSD}, choose the current-max-Q generating ϕ_k from $k\in\{1,2,...,K\}$ and the historical-steer-decided ϕ_k, respectively. The gradients of ϕ_k corresponding to these two selections are denoted as $\nabla_{\phi}^{\text{CMQ}}$ and $\nabla_{\phi}^{\text{HSD}}$, respectively, and can be presented in an uniform way

$$\nabla_{\phi_k}\pi^{\phi_k}=\mathbb{E}_{s_j,a_j\sim\mathscr{B}}\big[-Q^{\bar{\theta}}(s_j,a_j)\nabla_{\phi_k}\log\pi_{\phi_k}(a_j|s_j)\big], \quad (26)$$

where k is designated by $\arg\max_{\kappa}Q^{\bar{\theta}_{\kappa}}(s_j,a_j)$ or the stored historical-steer-decided k, respectively.

(2) The third chooses all ϕ_k from $k\in\{1,2,...,K\}$ to perform gradient back-propagation, letting all meta-policies learn from the same experience. The gradient of ϕ_k in this way is denoted as $\nabla_{\phi}^{\text{AMP}}$ and can be presented in the same form as Eq. (26) except that $k=1,2,...,K$.

*3) **Gradient Back-Propagation for Q Network**:* By virtue of Eq. (21), we know that $Q^{\theta}(s_j,a_j)$ and r_j are deterministic for a sampled e, but $\mathbb{E}_{a_{i+1}\sim\pi^{\bar{\phi}}}Q^{\bar{\theta}}(s_{i+1},a_{i+1})$ requires calculating expectation of $Q^{\bar{\theta}}(s_{i+1},a_{i+1})$ through sampling a_{j+1}. The elaboration comes from choosing the way to sample a_{j+1}. On account of the existence of multiple $\pi^{\phi_k},k\in\{1,2,...,K\}$ in FEAT, we can choose k in two manners to view $Q^{\bar{\theta}}(s_{i+1},a_{i+1})$ based on s_{j+1}: One is to select k via $\arg\max_k Q^{\bar{\theta}}(s_{j+1},a_{j+1}^k)$ where $a_{j+1}^k\sim\pi^{\phi_k}(a_{j+1}|s_{j+1})$, the other is to leave it to be decided by the current steer via $\omega^{\psi}(k|s_{j+1},\varepsilon)$ where ε is the environmental state stored along with e during exploration.

The loss functions of Q in these two manners are denoted by L_{θ}^{CMQ} and L_{θ}^{CSD}, respectively, and can be formulated by extending Eq. (21) in a uniform presentation as follows

$$L_{Q^{\theta}}=\mathbb{E}_{(s_j,a_j,r_j,s_{j+1})\sim\mathscr{B}}||Q^{\theta}(s_j,a_j)-(r_j+\gamma\max_{a_{j+1}\sim\pi^{\bar{\phi}}}Q^{\bar{\theta}}(s_{j+1},a_{j+1}^k))||_2, \quad (27)$$

[3]To consist with the form in existing work, ε is not explicitly shown in e.

[4]In this case, the k sampled from $z^{\psi}(k|\cdot)$ should also be included in ε.

where $k=\arg\max_{k'}Q^{\bar{\theta}}(s_{j+1},a_{j+1}^{k'})$ for L_{θ}^{CMQ} and $k\sim\omega^{\psi}(k|s_{j+1},\varepsilon)$ for L_{θ}^{CSD}. After determining k, the a_{j+1}^k could be sampled multiple times from $\pi^{\bar{\phi}_k}(a_{j+1}|s_{j+1})$, followed by the judgment of $\max_{a_{j+1}\sim\pi^{\bar{\phi}_k}}Q^{\bar{\theta}}(s_{j+1},a_{j+1}^k)$.

Here, it is worthy of noting that the stored k in e (not explicitly included for brevity) is not viewed as an appropriate option to sample a_{j+1}^k, since the stored k was steered by ω^{ψ} based on s_j rather than s_{j+1} during exploration.

We present the training algorithm of FEAT in Algorithm 1.

Algorithm 1 The training algorithm of FEAT.

```
Initialize θ, φ, ψ
Initialize target parameters θ̄ ← θ, φ̄ ← φ
Initialize experience buffer 𝒟
repeat
    Randomly select a group of values Λ, B, f^MD, f^ES
    Generate an environmental state ε ∈ Env
    Initialize Environment based on ε
    for each slot i do
        Observe the time-varying state s_i
        Sample k from ω^ψ(k|s_i, ε)
        Sample an action a_i from π^{φ_k}(a_i|s_i)
        Interact with Environment and receive s_{i+1} and r_i
        Store (s_i, a_i, r_i, s_{i+1}, k, ε) into 𝒟
    end for
    if perform learning then
        Sample a batch of (s_j, a_j, r_j, s_{j+1}, k, ε) from 𝒟
        Compute ∇_ψ ω, ∇_θ Q based on Eqs. (25) and (27)
        Update θ, φ, ψ based on ∇_θ Q, ∇_{φ_k} π^{φ_k}, ∇_ψ ω:
        θ=θ−λ_θ ∇_θ Q,  φ_k=φ_k−λ_φ ∇_{φ_k} π^{φ_k},  ψ=ψ−λ_ψ ∇_ψ ω
    end if
    Periodically update θ̄ and φ̄
until convergence
return φ, ψ
```

IV. Performance Evaluation

We develop a MEC simulator using Python, with the number of MDs, ES computing capacity, bandwidth, task arrival distribution being configurable for investigating environment adaptability of our and other state-of-the-art TOPA approaches. On one hand concerning MEC settings, there is one BS (empowered by an ES) and multiple MDs that are randomly distributed around BS from 20 m to 100 m, the main settings of which are listed in Table I unless otherwise specified. On the other hand concerning DRL-related settings: (1) The shared layers of π^k and hidden layers of Q^{θ} in Fig. 3 are both implemented using two fully-connected layers each of which is composed of 256 neuron nodes; (2) The discount factor γ w.r.t. future rewards is empirically set as 0.95, and the batch size of experience sampling is 64; (3) The temperature factor ζ in SAC is auto-tuned with the goal of constraining the expected entropy over visited states as in [30], and the Gumbel-Softmax temperature τ is annealed from 0.2. Throughout the evaluation, the TOPA performance is primarily investigated based on the metric $\frac{1}{N}\sum_{i=1}^{N}C_i$, i.e., the average cost of all MDs over the slots in an episode, where C_i is defined in Eq. (12).

Authorized licensed use limited to: Zhejiang University of Technology. Downloaded on September 23,2024 at 12:03:53 UTC from IEEE Xplore. Restrictions apply.

TABLE I
SIMULATION PARAMETERS

Parameter	Description	Value
δ	Time slot length	0.5 s
N	Number of time slots	10
c	CPU cycles per bit of task	[800, 900]
$t^{\max}$	Maximum tolerable latency	[0.8, 0.9] s
p^{MD}	Maximum transmitting power	0.5 W
f^{MD}	MD's computing capacity	1 GHz
eb^{MD}	MD's energy budget	8 J
ξ	Energy efficiency coefficient	10^{-27}
σ^2_{SINR}	Additive noise power	-100 dBm

A. Evaluation on the Training Components of FEAT

Distinct training components (w.r.t. the steer, meta-policy and Q network) have been discussed in Section III-C. Here, we will evaluate the performances of distinct combinations of the training components (training 20000 episodes with $M = 12$, $f^{\text{ES}} = 9$ GHz, $B = 10$ MHz), as shown in Table II. It can be observed that the 5-th combination, i.e., L_ψ^{CMQ}, ∇_ϕ^{HSD} and L_θ^{CMQ}, achieves the most promising performance both in converging speed (at the 1250-th episode) and converged cost (0.468). The combinations 1-2 and 6-12 are also seen to converge to similar costs (a bit higher than 0.468), with diverse converging speeds lower than the combination 5. However, the cost of the combinations 3 and 4 are found to be larger than 1, suggesting the unfavorable partnership between L_ψ^{HSD} and ∇_ϕ^{CMQ}. Hence, we will adopt the combination 5 in the following experiments.

TABLE II
EVALUATION OF THE TRAINING COMPONENTS OF FEAT

ID	Steer	Meta-Policy	Q Network	Converging Episode	Converging to the Cost
1	L_ψ^{CMQ}	∇_ϕ^{CMQ}	L_θ^{CMQ}	14500	0.481
2	L_ψ^{CMQ}	∇_ϕ^{CMQ}	L_θ^{CSD}	19000	0.469
3	L_ψ^{HSD}	∇_ϕ^{CMQ}	L_θ^{CMQ}	12500	1.034
4	L_ψ^{HSD}	∇_ϕ^{CMQ}	L_θ^{CSD}	16000	1.181
5	L_ψ^{CMQ}	∇_ϕ^{HSD}	L_θ^{CMQ}	**1250**	**0.468**
6	L_ψ^{CMQ}	∇_ϕ^{HSD}	L_θ^{CSD}	5000	0.469
7	L_ψ^{HSD}	∇_ϕ^{HSD}	L_θ^{CMQ}	3000	0.469
8	L_ψ^{HSD}	∇_ϕ^{HSD}	L_θ^{CSD}	9500	0.478
9	L_ψ^{CMQ}	∇_ϕ^{AMP}	L_θ^{CMQ}	1650	0.469
10	L_ψ^{CMQ}	∇_ϕ^{AMP}	L_θ^{CSD}	4300	0.479
11	L_ψ^{HSD}	∇_ϕ^{AMP}	L_θ^{CMQ}	2100	0.470
12	L_ψ^{HSD}	∇_ϕ^{AMP}	L_θ^{CSD}	7000	0.481

B. Evaluation on the Number of Meta-Policies

In Fig. 4, we further evaluate the performance of FEAT w.r.t. different numbers of meta-policies K. As shown, FEAT is found to suffer from obvious instability during training, when $K = 1$ or $K = 5$. Besides, it could be seen that FEAT converges slowly when $K = 7$ (converging after 5000 episodes) and $K = 9$ (converging after 15000 episodes). Among the investigated numbers of K, FEAT is noticed to achieve the best performance when $K = 3$, converging at the episode around 1350 to the cost 0.46. With the observation, we will set $K = 3$ in the following experiments.

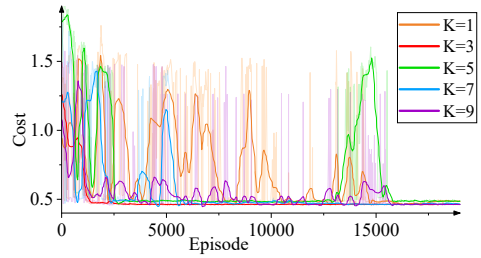


Fig. 4. Performance evaluation w.r.t. the numbers of meta-policies in FEAT.

C. Analysis on FEAT via Visualization

In this part, we analyze the advantages brought from FEAT through visualizing TOPA states and actions.

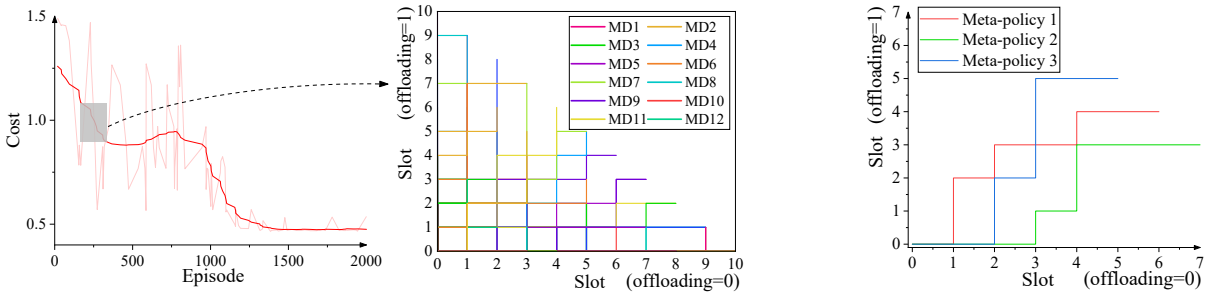
(1) In Fig. 5(a), offloading actions in the initial 100 training episodes of FEAT are plotted, with the horizontal and vertical axes indicating non-offloading and offloading decisions, respectively. It can be seen that the offloading action paths over the 100 episodes spread all over the positive horizontal and vertical coordinates, implying that FEAT is attempting to explore the offloading action space as much as possible during the initial training stages which would be of great help for FEAT to achieve high converging speeds (as shown by the 5-th combination in Table II and the $K = 3$ curve in Fig. 4).

(2) In Fig. 5(b), offloading decisions of MD^2 from three meta-policies over an episode are taken as a visualization example. In slot one, both three meta-policies are observed to choose a horizontal step (local computing), i.e., $a_1^2 = 0$ for $\pi^{\phi_k}, \forall k \in \{1, 2, 3\}$. Then, in slot two, π^{ϕ_1} is seen to choose a vertical step (edge computing) for MD^2, and π^{ϕ_2} and π^{ϕ_3} are still found to choose horizontal steps. From the third to the last slot, it can be noticed that π^{ϕ_1}, π^{ϕ_2} and π^{ϕ_3} choose distinct horizontal/vertical steps for MD^2. The offloading decisions of $\pi^{\phi_k}, \forall k \in \{1, 2, 3\}$ for other MDs are also found distinctive over an episode, and so is it in other episodes (not shown due to the space limitation). It demonstrates that the meta-policies of FEAT indeed play different TOPA roles, the ability of which is automatically obtained during experience learning.

(3) In Fig. 6, we adopt t-Distributed Stochastic Neighbor Embedding (t-SNE) [33], a statistical method for visualizing high-dimensional data by giving each data-point a location in a two-dimensional map, to cluster the actions taken by FEAT on 1K states (along with ε) randomly sampled after training convergence. When coloring the three clusters with different colors, it can be found that the number of colors equals that of meta-policies ($K = 3$), meaning that the steerer ω^ψ can choose a meta-policy for a cluster of states and ε (in line with the expectation of FEAT during policy design in Section III-A).

(4) Furthermore, we visualize the values of $\omega^\psi(k|\cdot)$, $\pi^{\phi_k}, \forall k \in \{1, 2, 3\}$ and s_i in the first slot from episode 110 to 200, as shown in Fig. 7. In the top row, $\omega^\psi(k|\cdot)$ is found to assign different weights for each meta-policy $k \in \{1, 2, 3\}$, illustrating distinct probabilities for choosing meta-policies (whose actions are shown in the middle row) according to s_i

Authorized licensed use limited to: Zhejiang University of Technology. Downloaded on September 23,2024 at 12:03:53 UTC from IEEE Xplore. Restrictions apply.



(a) Exploration visualization w.r.t. offloading actions (12 MDs with each being plotted using one color) during the initial training stages (200-300 episodes).

(b) Offloading action visualization of meta-policies during an episode.

Fig. 5. Visualization of offloading actions during (a) the initial training stages and (b) an episode after convergence.

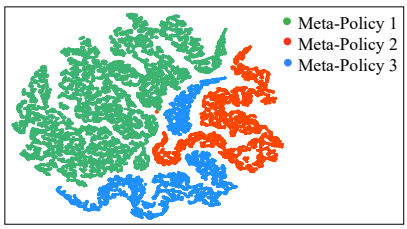


Fig. 6. Cluster the TOPA actions taken by FEAT on 1 K states.

(the bottom row) and environmental state ε. The choice of meta-policies by $\omega^{\psi}(k|\cdot)$ is observed to strongly relate to s_i, with the meta-policy 1 being more active from episode 110 to 140 while the meta-policy 3 more active from episode 180 to 200. This further verifies that different meta-policies are good at making TOPA decisions in distinct clusters of internal states.

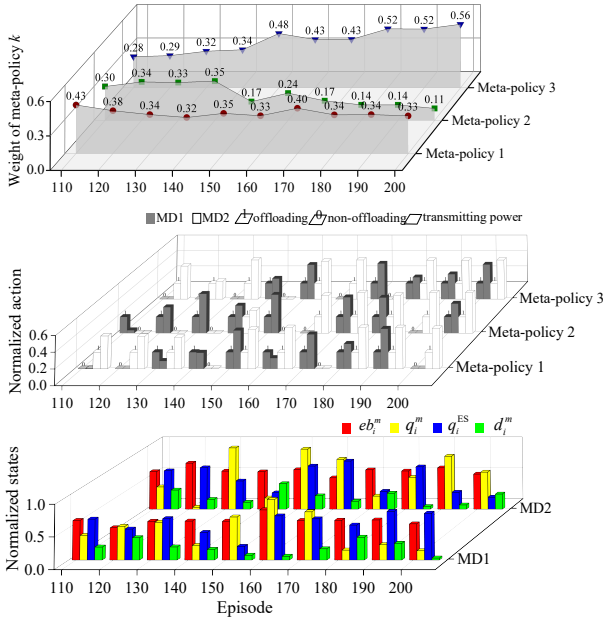


Fig. 7. Visualization of s_i, π^{ϕ_k} and $\omega^{\psi}(k|\cdot)$ from episode 110 to 200.

D. Performance Comparison between FEAT and State-of-the-art DRL-based TOPA Approaches

Fig. 8 shows the training process of TOPA based on DDPG, TD3, SAC, FEAT in a set of environments where $B\in\{9,10,11\}$ MHz, $f^{\text{ES}}\in\{8.5,9,9.5\}$ GHz, $\Lambda=\mathcal{N}(700,\sigma^2)$ Kb where $\sigma^2\in\{10^3, 5\times10^3, 1.5\times10^4, 5\times10^4, 10^5\}$ (Λ, an element of ε defined in Eq. (11), is the arrival distribution of d_i^m). FEAT and SAC based approaches are found to achieve similar converging speed (about 5000 episodes) and converged cost (near to 0.46), while TD3 and DDPG are observed to be more unstable (achieving convergence after 8500 and 17500 episodes, respectively).

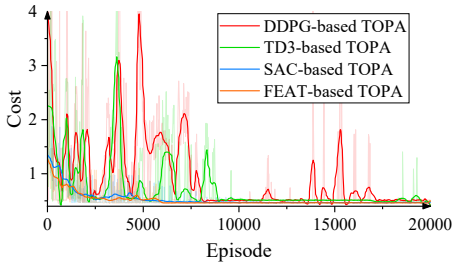


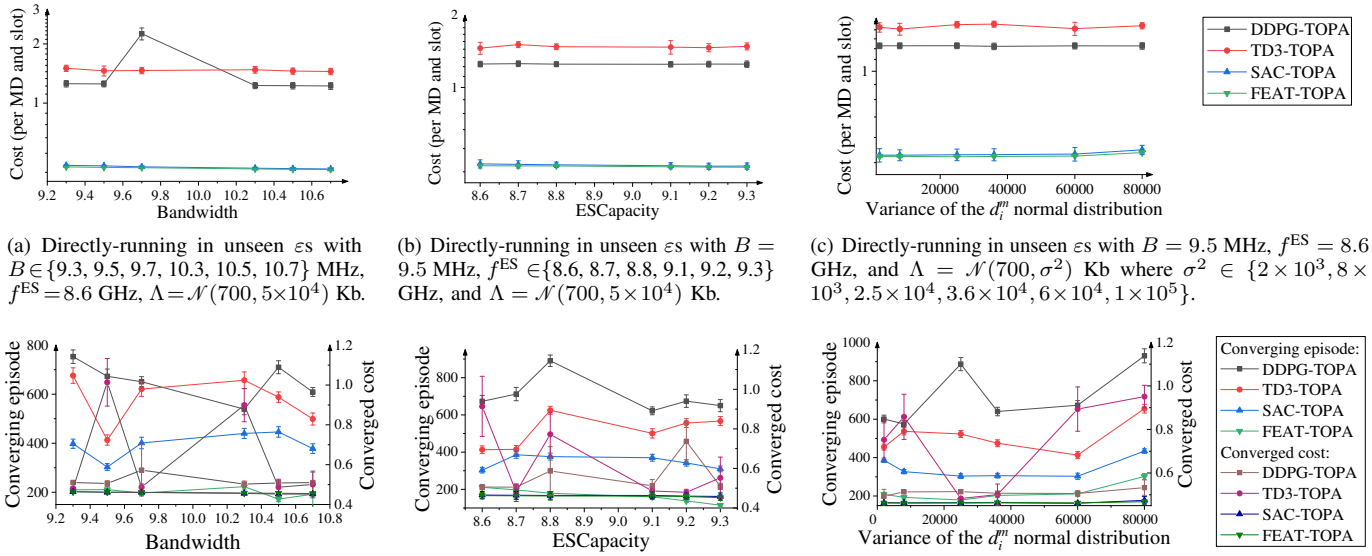
Fig. 8. Training comparison among different approaches.

In Fig. 9, the performances of four trained approaches are compared in directly-running and fine-tuning, respectively. **(1)** When applied directly in unseen εs as shown in Figs. 9(a) to 9(c), DDPG and TD3 based TOPA are found to suffer from obvious cost degradation (consistent with the preliminary experiments in Fig. 1(b)) while FEAT could still achieve the lowest cost (around 0.48) comparable to the converged value in Fig. 8. **(2)** When allowed to fine-tune in unseen εs as shown in Figs. 9(d) to 9(f), FEAT is also observed to achieve the highest converging speed (around 200 episodes, while others are higher than 300 episodes) and lowest converged cost (about 0.47). Additionally, it can be noticed that the fine-tuning difficulty generally declines with the rise of bandwidth or ES capacity while grows with the variance of d_i^m normal distribution.

E. Performance Evaluation on a Testbed

Moreover, we build a real TOPA testbed (as shown in Fig. 10) similar to [34], that consists of a WiFi router (802.11n wireless LAN), a desktop equipped with Intel Core i9-11900K (8 cores, each 3.5 GHz) and 32 GB RAM, and four Raspberry Pi 4 Bs (4 cores, 1.5 GHz, 4GB RAM) and XTark robotic vehicles (equipped with a Raspberry Pi 4 B per vehicle), playing the roles of the BS, ES, and MDs, respectively. Face recognition, one of the most popular mobile applications, is taken as the

Authorized licensed use limited to: Zhejiang University of Technology. Downloaded on September 23,2024 at 12:03:53 UTC from IEEE Xplore. Restrictions apply.



(a) Directly-running in unseen εs with $B \in \{9.3, 9.5, 9.7, 10.3, 10.5, 10.7\}$ MHz, $f^{\text{ES}} = 8.6$ GHz, $\Lambda = \mathcal{N}(700, 5\times10^4)$ Kb.

(b) Directly-running in unseen εs with $B = 9.5$ MHz, $f^{\text{ES}} \in \{8.6, 8.7, 8.8, 9.1, 9.2, 9.3\}$ GHz, and $\Lambda = \mathcal{N}(700, 5\times10^4)$ Kb.

(c) Directly-running in unseen εs with $B = 9.5$ MHz, $f^{\text{ES}} = 8.6$ GHz, and $\Lambda = \mathcal{N}(700, \sigma^2)$ Kb where $\sigma^2 \in \{2\times10^3, 8\times10^3, 2.5\times10^4, 3.6\times10^4, 6\times10^4, 1\times10^5\}$.

(d) Fine-tuning in unseen εs with $B \in \{9.3, 9.5, 9.7, 10.3, 10.5, 10.7\}$ MHz, $f^{\text{ES}} = 8.6$ GHz, $\Lambda = \mathcal{N}(700, 5\times10^4)$ Kb.

(e) Fine-tuning in unseen εs with $B = 9.5$ MHz, $f^{\text{ES}} \in \{8.6, 8.7, 8.8, 9.1, 9.2, 9.3\}$ GHz, and $\Lambda = \mathcal{N}(700, 5\times10^4)$ Kb.

(f) Fine-tuning in unseen εs with $B = 9.5$ MHz, $f^{\text{ES}} = 8.6$ GHz, and $\Lambda = \mathcal{N}(700, \sigma^2)$ Kb where $\sigma^2 \in \{2\times10^3, 8\times10^3, 2.5\times10^4, 3.6\times10^4, 6\times10^4, 8\times10^4\}$.

Fig. 9. Performance comparison between FEAT and state-of-the-art DRL-based TOPA approaches in directly-running and fine-tuning.

MD's task and includes three types of face picture sizes (10+ KB, 30+ KB, and 60+ KB).

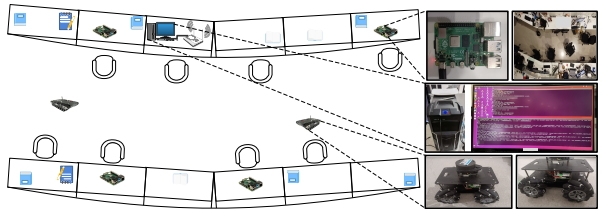
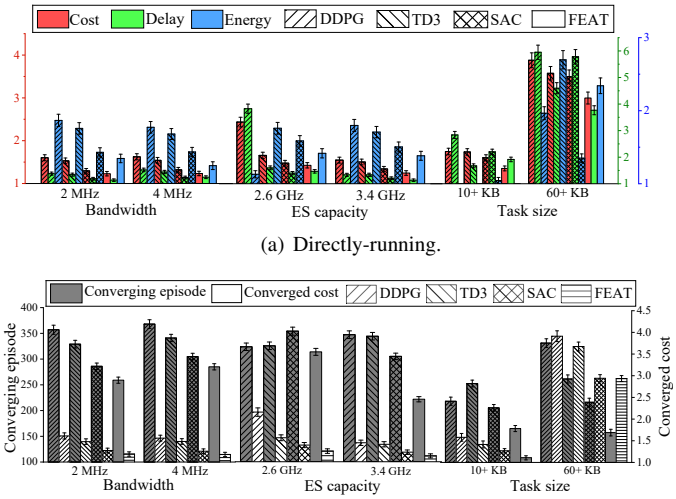


Fig. 10. A TOPA testbed implemented with the WiFi router, desktop, Raspberry models and robotic vehicles.

The experiments are conducted with diverse WiFi bandwidths, desktop computing frequencies, and face picture sizes. Concretely speaking, DDPG, TD3, SAC and FEAT based TOPA approaches are trained with 3 MHz bandwidth, one desktop core running at 3 GHz, and 30+ KB picture size, the converged models from which directly-run and fine-tune in the environments with $\{2, 4\}$ MHz bandwidths, one desktop core running at $\{2.6, 3.4\}$ GHz, and $\{10+, 60+\}$ KB picture sizes. To gain a more intuitive view on the experiments, we record both the task latency and energy consumption (per MD and slot) and plot them along with the cost in Fig. 11 under distinct environments. It can be observed that FEAT achieves the lowest costs (1.22, 1.23, 1.41, 1.24, 1.36, 3.05) and energy-consumption/task-latency (1.12 s, 1.24 J, 1.39 J, 1.12 s, 0.82 J, 3.76 J) during directly-running as shown in Fig. 11(a), and also the fastest converging speed (259, 285, 318, 224, 167, 159 episodes) during fine-tuning as shown in Fig. 11(b), when $B = 2/4$ MHz, $f^{\text{ES}} = 2.6/3.4$ GHz, $d_i^m = 10/60+$ KB, respectively. FEAT achieves approximately 28.1%, 24.4% and 16.4% performance improvements against DDPG, TD3 and SAC based approaches, respectively, w.r.t. converging episode.



(a) Directly-running.

(b) Fine-tuning.

Fig. 11. Performance comparison in the testbed with distinct environments.

V. Conclusion

In this paper, we propose FEAT to enable fast environment-adaptive task offloading and power allocation in MEC. The key components of FEAT include an environment-embedded steerer and a group of ensemble reinforcement learning-based meta-policies. The steerer and meta-policies are both trained in an end-to-end manner in a diverse set of environments, and only the steerer needs a small amount of fine-tuning when encountering unseen environments. Extensive experimental results conducted on both simulation and testbed demonstrate the superiority of FEAT in terms of environment adaptation against the state-of-the-art DRL-based TOPA approaches.

Authorized licensed use limited to: Zhejiang University of Technology. Downloaded on September 23,2024 at 12:03:53 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] J. Navarro-Ortiz, P. Romero-Diaz, S. Sendra, P. Ameigeiras, J. J. Ramos-Munoz, and J. M. Lopez-Soler, "A survey on 5g usage scenarios and traffic models," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 905–929, 2020.

[2] P. Porambage, J. Okwuibe, M. Liyanage, M. Ylianttila, and T. Taleb, "Survey on multi-access edge computing for internet of things realization," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 2961–2991, 2018.

[3] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, 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] A. Shakarami, M. Ghobaei-Arani, and A. Shahidinejad, "A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective," *Computer Networks*, vol. 182, p. 107496, 2020.

[6] T. T. Vu, N. Van Huynh, D. T. Hoang, D. N. Nguyen, and E. Dutkiewicz, "Offloading energy efficiency with delay constraint for cooperative mobile edge computing networks," in *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2018, pp. 1–6.

[7] J. Luo, X. Deng, H. Zhang, and H. Qi, "Qoe-driven computation offloading for edge computing," *Journal of Systems Architecture*, vol. 97, pp. 34–39, 2019.

[8] H. Guo and J. Liu, "Collaborative computation offloading for multiaccess edge computing over fiber–wireless networks," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 5, pp. 4514–4526, 2018.

[9] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 587–597, 2018.

[10] F. Zhou, Y. Wu, R. Q. Hu, and Y. Qian, "Computation rate maximization in uav-enabled wireless-powered mobile-edge computing systems," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 9, pp. 1927–1941, 2018.

[11] 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 Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.

[12] T. K. Rodrigues, K. Suto, H. Nishiyama, J. Liu, and N. Kato, "Machine learning meets computation and communication control in evolving edge and cloud: Challenges and future perspective," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 1, pp. 38–67, 2019.

[13] M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu, and W. Zhuang, "Learning-based computation offloading for iot devices with energy harvesting," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 2, pp. 1930–1941, 2019.

[14] T. P. Truong, T.-V. Nguyen, W. Noh, S. Cho *et al.*, "Partial computation offloading in noma-assisted mobile-edge computing systems using deep reinforcement learning," *IEEE Internet of Things Journal*, vol. 8, no. 17, pp. 13 196–13 208, 2021.

[15] X. Qiu, W. Zhang, W. Chen, and Z. Zheng, "Distributed and collective deep reinforcement learning for computation offloading: A practical perspective," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 5, pp. 1085–1101, 2020.

[16] N. Naderializadeh, J. J. Sydir, M. Simsek, and H. Nikopour, "Resource management in wireless networks via multi-agent deep reinforcement learning," *IEEE Transactions on Wireless Communications*, vol. 20, no. 6, pp. 3507–3523, 2021.

[17] 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 Transactions on Wireless Communications*, vol. 20, no. 11, pp. 7519–7537, 2021.

[18] G. Qu, H. Wu, R. Li, and P. Jiao, "Dmro: A deep meta reinforcement learning-based task offloading framework for edge-cloud computing," *IEEE Transactions on Network and Service Management*, 2021.

[19] 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 Transactions on Wireless Communications*, vol. 20, no. 3, pp. 1487–1500, 2020.

[20] X. Pang, Z. Wang, J. Li, R. Zhou, J. Ren, and Z. Li, "Towards online privacy-preserving computation offloading in mobile edge computing," in *2022 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2022, pp. 1179–1188.

[21] X. Zhou, L. Huang, T. Ye, and W. Sun, "Computation bits maximization in uav-assisted mec networks with fairness constraint," *IEEE Internet of Things Journal*, 2022.

[22] T. Zhu, J. Li, Z. Cai, Y. Li, and H. Gao, "Computation scheduling for wireless powered mobile edge computing networks," in *2020 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2020, pp. 596–605.

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

[24] T. Van Chien, C. Mollén, and E. Björnson, "Large-scale-fading decoding in cellular massive mimo systems with spatially correlated channels," *IEEE Transactions on Communications*, vol. 67, no. 4, pp. 2746–2762, 2018.

[25] Y. Ding, K. Li, C. Liu, and K. Li, "A potential game theoretic approach to computation offloading strategy optimization in end-edge-cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 6, pp. 1503–1519, 2021.

[26] Y. Chen, Z. Liu, Y. Zhang, Y. Wu, X. Chen, and L. Zhao, "Deep reinforcement learning-based dynamic resource management for mobile edge computing in industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 7, pp. 4925–4934, 2020.

[27] 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 *2022 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2022, pp. 1009–1018.

[28] 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.

[29] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

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

[31] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," *arXiv preprint arXiv:1611.00712*, 2016.

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

[33] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne." *Journal of machine learning research*, vol. 9, no. 11, 2008.

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

Authorized licensed use limited to: Zhejiang University of Technology. Downloaded on September 23,2024 at 12:03:53 UTC from IEEE Xplore. Restrictions apply.