Appendix A
Differences in Generalization Compared to Related Works

We summarize the distinctions between our approach and existing works in Table A1. Many studies [1]–[6] are limited to problems that optimize for a single preference. Li te al. [3] employ a Q-learning-based deep reinforcement learning (DRL) method to solve the computation offloading problem in a multi-user environment. Cui et al. [1] decomposes user association, offloading decision, computing, and communication resource allocation into two related sub-problems and employs the DQN algorithm for decision-making. Lei et al. [2] proposed a DRL-based joint computation offloading and multi-user scheduling algorithm for IoT edge computing systems, aiming to minimize the long-term weighted sum of delay and power consumption under stochastic traffic arrivals. Huang et al. [4] employed an improved DQN method to address offloading decision problems and resource allocation problems. The above works focus on two objectives, delay and energy consumption, and use a weight coefficient to balance them or optimize one objective while satisfying the constraints of the other. Moreover, these studies lack research on the generalization.

Some studies [7]–[11] focus only on the generalization of system parameters. Li et al. [7] combine graph neural networks and seq2seq networks to make decisions on task offloading. They employ a meta-reinforcement learning approach to enhance the generalization of the offloading strategy in environments with different system parameters. Ren et al. [8] design a set of experience maintaining and sampling strategies to improve the training process of DRL, enhancing the model's generalization to different environments. Wang et al. [9] design an offloading decision algorithm based on meta-reinforcement learning, which uses a seq2seq neural network to represent the offloading policy. This approach can adapt to various environments covering a wide range of topologies, task numbers, and transmission rates. Wu et al. [10] propose a method that combines graph neural networks and DRL, which can be applied to various environments with inter-dependencies among different tasks. Hu et al. [11] propose a size-adaptive offloading scheme and a setting-adaptive offloading component, designed to quickly adapt to new MEC environments of varying sizes and configurations with a few interaction steps. The above work only considers generalization in terms of system parameters, without addressing generalization in terms of the number of servers and multi-preference issues.

Other works [12]–[14] only consider the generalization of the number of servers. A few works consider the generalization of both system parameters and the number of servers. Gao et al. [15] model the decentralized task offloading problem as a partially observable Markov decision process and use a multi-agent RL method to train the policy. They consider the generalization of both system parameters and the number of servers, but do not explore multi-preference issues. Our method provides a deeper exploration of the generalization of the offloading strategy, considering the generalization in terms of multi-preference, system parameters, and server quantities.

Appendix B
Supplementary Figures

A. *System Model*

The MEC system model we consider is illustrated in Fig. A1. An MEC system consists of E edge servers, one remote cloud server. The system processes M tasks arriving sequentially, with each task being uploaded to only one server.

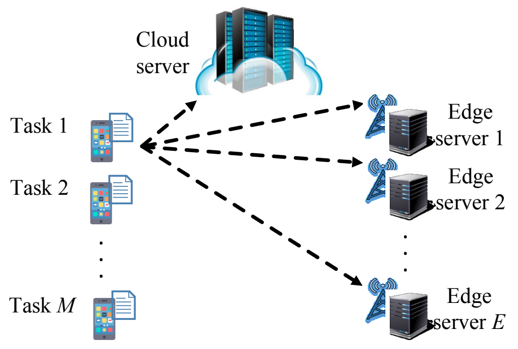


Fig. A1: An illustrative example system model of MEC.

The problem (10) is non-convex due to constraint (10b), which requires the decision variables to be discrete. This makes the feasible set non-convex, as linear combinations of feasible solutions are not guaranteed to remain feasible, leading to the non-convex nature of the problem. Moreover, when making offloading decisions at each time step, the sizes of tasks arriving after that time step are unknown. As shown in Eq. (5), (6), and (7), the execution time of a task is related to the offloading decisions made in subsequent time steps, as well as the size of the tasks. Therefore, without information about future time steps, convex optimization methods cannot be used to solve problem (10).

The challenge of this problem lies in two aspects: First, there is a conflict between optimizing delay and energy consumption. According to Eq. (4) and Eq. (8), the main energy consumption of a task depends on execution energy, which increases with higher server CPU frequencies. Therefore, reducing energy consumption involves offloading tasks to edge servers with lower CPU frequencies. According to Eq. (3) and Eq. (7), the main delay of a task depends on execution time, which is lower on cloud servers with higher CPU frequencies, but increases as more tasks are executed on a single server. Thus, reducing delay requires offloading a larger number of tasks to cloud servers with higher CPU frequencies, leading to a conflict between optimizing delay and energy consumption. Second, the scheduling policy must optimize problem (10) under distinct preferences to achieve the optimal solution, rather than just under a fixed preference.

B. *GMORL Algorithm*

The structure of the GMORL algorithm is illustrated in Fig. A2. During a single episode, the contextual MOMDP samples a context c from the context space $\mathcal{C}$ to create an environment. The agent observes the state s_t, which includes the context c, and feeds it into a policy neural network. The

TABLE A1
RELATE WORKS ABOUT DRL METHOD FOR OFFLOADING TASK SCHEDULING IN MEC SYSTEM.

Refs.	Generalization across different aspects		
	Multi-preference	System parameters	Server quantities
[1]–[6]	✘	✘	✘
[7]–[11]	✘	✔	✘
[12]–[14]	✘	✘	✔
[15]	✘	✔	✔
Ours	✔	✔	✔

policy network's output determines the action a_t, which the agent takes to offload a task, receiving a scalarized reward $r_{\boldsymbol{\omega}}(\boldsymbol{s}_t, a_t)$ in return. Subsequently, the contextual MOMDP transitions to the next state $\boldsymbol{s}_{t+1}$.

The experience tuple $\langle \boldsymbol{s}_t, a_t, r_{\boldsymbol{\omega}}(\boldsymbol{s}_t, a_t), \boldsymbol{s}_{t+1} \rangle$ is stored in a replay buffer. After multiple episodes, the replay buffer accumulates a diverse set of experiences covering the context space $\mathcal{C}$. During each training phase, samples from the replay buffer are used to update the neural networks. This process enables the agent to learn the optimal policy for any given context $c \in \mathcal{C}$.

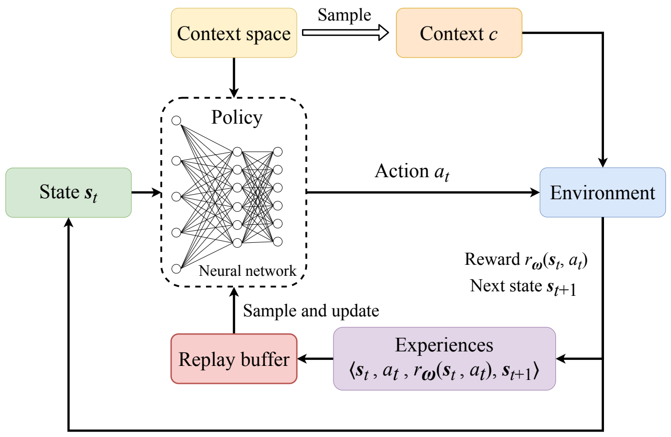


Fig. A2: The overview of the GMORL algorithm.

C. The Neural Network Architecture

The neural network takes the state information of each server and the context as input, processes the features of each server individually through convolutional modules, then aggregates all features through MLP modules, and finally, for the actor network, outputs the selection probabilities for each server, and for the critic network, outputs the values of each server.

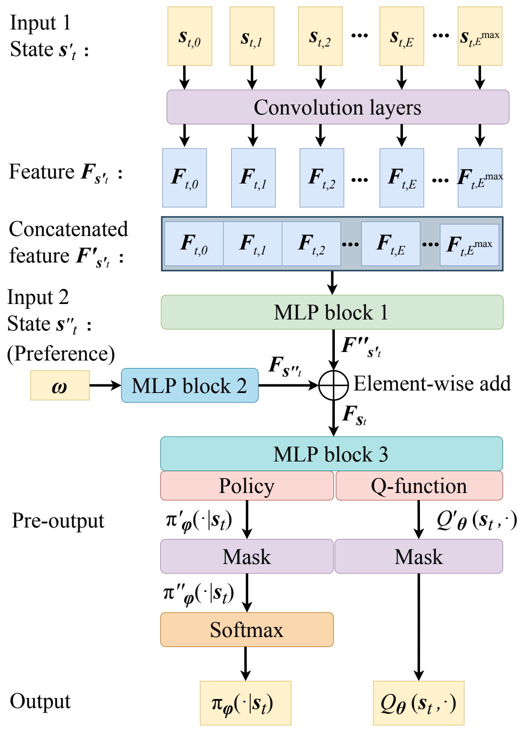


Fig. A3: The neural network architecture of the scheduling policy.

D. The ascent Simplex

As shown in Fig. A4, the green and blue arrows denote the gradient directions of the delay and energy consumption objectives, respectively. The light blue area stands for an ascent simplex.

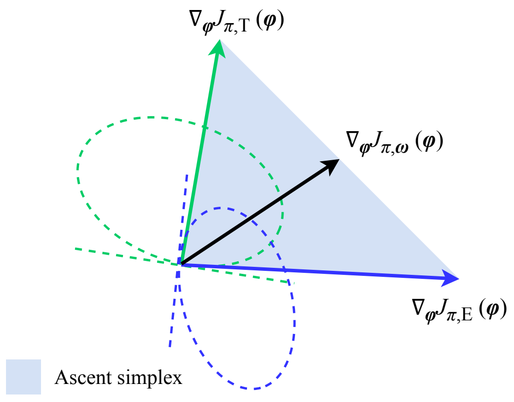
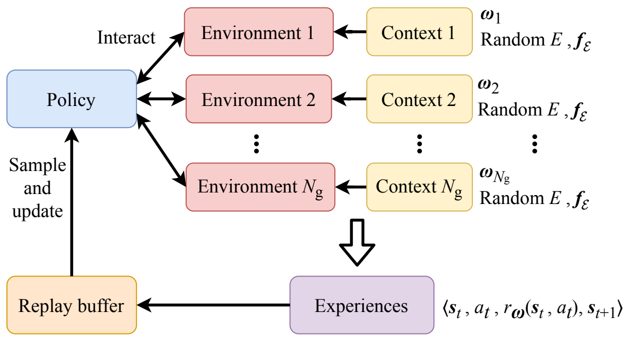

$\nabla_\varphi J_{\pi,\mathrm{T}}(\varphi)$
$\nabla_\varphi J_{\pi,\omega}(\varphi)$
$\nabla_\varphi J_{\pi,\mathrm{E}}(\varphi)$
Ascent simplex

Fig. A4: The ascent simplex in a 2–objectives problem.

When the gradient directions of the two objectives are not completely opposite, the ascent simplex exists. If the gradient direction lies within the scent simplex, both objectives can be optimized simultaneously, and the gradient descent algorithm can reach a Pareto local optimum.

E. Learning Approach

During the training phase, we sample N_{g} contexts to create N_{g} MEC environments for each epoch. The preferences of these environments are determined by Eq. (32), while their number of servers E and frequencies $\boldsymbol{f}_{\mathcal{E}}$ are randomly drawn from the context space. These environments interact with the policy to generate experiences, which are stored in the replay buffer and used to update the policy.



Interact
Policy
Sample and update
Environment 1
Environment 2
Environment N_{g}
Context 1
Context 2
Context N_{g}
ω_1 Random E , $f_{\mathcal{E}}$
ω_2 Random E , $f_{\mathcal{E}}$
$\omega_{N_{\mathrm{g}}}$ Random E , $f_{\mathcal{E}}$
Replay buffer
Experiences
$\langle s_t, a_t, r_{\omega}(s_t, a_t), s_{t+1}\rangle$

Fig. A5: The generalization learning approach.

Appendix C
Simulation Setup

We present the detailed listing of our model in Table A2 and provide the context in Table A3. We set testing preference set $\Omega_{N_{\mathrm{g}}}$ according to Eq. (32) and fit Pareto front in N_{g} preferences. Each preference's performance contains total delay and energy consumption for all tasks in one episode. We evaluate a performance (delay or energy consumption) with an average of 1000 episodes. A disk coverage has a radius of 1000m to 2000m for a cloud server and 50m to 500m for an edge server. Each episode needs to initial different radiuses for the cloud and edge servers. We set the mean of task size $\bar{L}$ according to Eq. (1).

A. Evaluation Metrics

We consider the following metrics to evaluate the performances of the proposed algorithms.

- **Energy Consumption:** The total energy consumption of one episode given as $\sum_{m=1}^{M} E_m^{\mathrm{off}} + E_m^{\mathrm{exe}}$, and the average energy consumption per Mbits task of one episode given by $\sum_{m=1}^{M} \frac{E_m^{\mathrm{off}} + E_m^{\mathrm{exe}}}{\bar{L}}$.
- **Task Delay:** The total energy consumption of one episode given as $\sum_{m=1}^{M} E_m^{\mathrm{off}} + E_m^{\mathrm{exe}}$, and the average energy consumption per Mbits task of one episode given by $\sum_{m=1}^{M} \frac{E_m^{\mathrm{off}} + E_m^{\mathrm{exe}}}{\bar{L}}$.

TABLE A2
Model Parameters

Resource Scheduling Hyperparameters	**Values**
The number of steps for one episode T	100
Step duration Δt	1 s
The number of users U	10
The number of tasks M	100
System bandwidth W	16.6MHz [16]
Offloading power p^{off}	10 mW
The number of CPU cycles η for one-bit task	10^3
Effective capacitance coefficient κ	5×10^{-31}
Poisson arrival rate λ_p for each user	0.1
DRL Hyperparameters	**Values**
The number of epochs for training N_{ep}	4000
The number of environments for one epoch N_{g}	64
Update round N_{up}	10
Replay memory	1×10^5
Batch size	4096
SAC temperature parameter α_H	0.05
The learning rate of policy λ_π	1×10^{-6}
The learning rate of soft Q-function λ_Q	1×10^{-6}
The learning rate of temperature λ_{α_H}	0
Discount factor γ	0.95

- **Pareto Front:** $PF(\Pi) = \{\pi \in \Pi \mid \nexists \pi' \in \Pi : \boldsymbol{y}^{\pi'} \succ_P \boldsymbol{y}^{\pi}\}$, where the symbols are defined by Eq. (12).
- **Hypervolume Metric:** $\mathcal{V}(PF(\Pi)) = \int_{\mathbb{R}^2} \mathbb{I}_{V_h(PF(\Pi))}(z)dz$, where the symbols are defined by Eq. (14).

Appendix D
Convergence Performances

We verify the convergence of the proposed GMORL algorithm. In Fig. A6a, we evaluate and plot the training reward of our algorithm. The reward shown in this figure is scalarized using Eq. (29). We observe that with the training episode increasing, the total reward converges. In fig. A6b and fig. A6c, as the training episodes increase, the delay and energy consumption decrease and converge to a stable value. This indicates that the GMORL algorithm converges effectively and reach a Pareto local optimum. In the following subsection, we will specifically analyze other performances in various system settings.

TABLE A3
CONTEXT SPACE FOR TRAINING AND TESTING

Context space	**Training**	**Testing**
The number of preference N_g	64	100
Edge server quantity $\mathcal{C}_E$	$\{1, 2, \ldots, 8\}$	$\{1, 2, \ldots, 10\}$
Cloud server CPU frequency $\mathcal{C}_{f_0}$	$[3.5, 4.5]$ GHz	$[3.0, 5.0]$ GHz
Edge server CPU frequency $\mathcal{C}_{\boldsymbol{f}_{\mathcal{E}'}}$	$[1.75, 2.25]$ GHz	$[1.5, 2.5]$ GHz

APPENDIX E
BASELINE INTRODUCTION

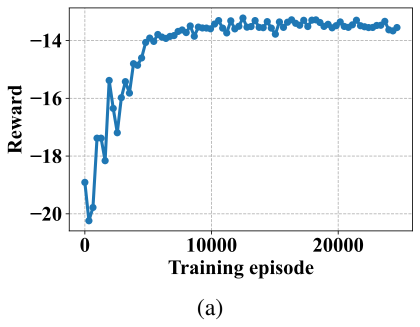
LinUCB-based scheme: The Offloading scheme is based on a kind of contextual MAB algorithm [17]. It is an improvement over the traditional UCB algorithm. This scheme uses states as MAB contexts and learns a policy by exploring different actions. We apply the multi-arm bandit algorithm. We regard each action as an arm and construct the feature of an arm from preference $\boldsymbol{\omega}$ and server information vector $\boldsymbol{s}_{t,e}$. Then, we update the parameter matrix based on the context and exploration results to learn a strategy that maximizes rewards. We train this scheme in preference set Ω_{101} and evaluate it for any preference in one. This method is computationally simple and incorporates context information, making it widely used in task offloading.

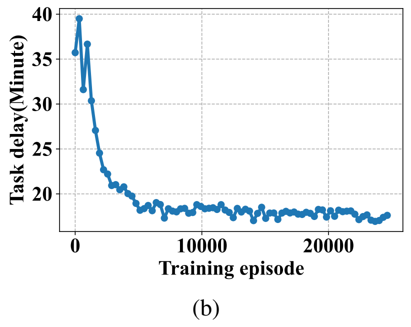
SA-based scheme: The heuristic method searches for an optimal local solution for task offloading without contexts. We use this method to observe the performance of heuristic approaches. This method generates a fixed offloading scheme for each preference and then iteratively searches for better solutions through local search. Once a better solution is found, it is accepted or rejected with a certain probability. This scheme searches 10000 episodes for each preference. However, searching for a solution that only applies to a specific context is time-consuming.

Random-based scheme: The random-based scheme has p probability to offload a task to the cloud server and $1-p$ probability to a random edge server. We tune the probability p and evaluate the scheme to obtain a Pareto front.

Multi-policy scheme: The multi-policy MORL approach [13] is based on the standard Discrete-SAC algorithm. We build 101 Discrete-SAC policy models for the 101 preference in Ω_{101} correspondingly. We train each policy model with $f_0 = 4$ GHz and $f_{e'} = 2$ GHz. This method has no generalization ability. A well-trained policy model is applicable to a specific context. However, benefiting from focusing on a specific context, this method is more likely to achieve optimal performance. We apply the method to determine the upper bound of the Pareto front.

APPENDIX F
DETAILS OF POLICY UPDATE





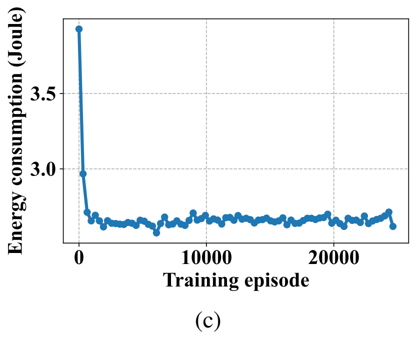


Fig. A6: Convergence performance of the proposed GMORL algorithm: (a) Reward during training; (b) Total delay during training when $E = 5$, $f_0 = 4$ GHz, $f_{e'} = 2$ GHz for all $e' \in \mathcal{E}'$, and $\boldsymbol{\omega} = (1, 0)$; (c) Total energy consumption during training when $E = 5$, CPU frequency $f_0 = 4$ GHz, $f_{e'} = 2$ GHz for all $e' \in \mathcal{E}'$, and preference $\boldsymbol{\omega} = (0, 1)$.

We present the proposed GMORL in Algorithm 1. The policy update process of GMORL is based on the Discrete-SAC algorithm, which includes a policy evaluation step and a policy improvement step. The Discrete-SAC algorithm augments the standard objective by an entropy term. The maximum entropy objective incentivizes a policy to explore more widely and captures multiple modes with near-optimal behavior. The optimal Discrete-SAC policy with maximum entropy objective is

$$\pi^* = \arg\max_{\pi} \sum_{t}^{T} \mathbb{E}_{(\boldsymbol{s}_t, a_t) \sim \rho_\pi} [\gamma^t (r_{\boldsymbol{\omega}}(\boldsymbol{s}_t, a_t) + \alpha_H \mathcal{H}(\pi(\cdot|\boldsymbol{s}_t)))], \tag{A1}$$

where ρ_π denotes the trajectory distribution of policy π, and α_H is a temperature parameter that determines the importance of the entropy term. The action probability vector of policy π at state $\boldsymbol{s}_t$ is $\pi(\cdot|\boldsymbol{s}_t)$. The entropy of $\pi(\cdot|\boldsymbol{s}_t)$ is $\mathcal{H}(\pi(\cdot|\boldsymbol{s}_t))$, and it satisfies $\mathcal{H}(\pi(\cdot|\boldsymbol{s}_t)) = -\log \pi(\cdot|\boldsymbol{s}_t)$.

In the policy evaluation step, we can obtain the soft Q-value

Algorithm 1 The GMORL Scheduling Algorithm

```
1: Initialize replay buffer $\mathcal{D}$, policy network parameters $\boldsymbol{\phi}$,
   the parameters of two local Q-function networks $\boldsymbol{\theta}_1$ and
   $\boldsymbol{\theta}_2$, the parameters of two target Q-function networks $\bar{\boldsymbol{\theta}}_1$
   and $\bar{\boldsymbol{\theta}}_2$.
2: Given training context space $\mathcal{C}$ and set preference context
   space $\Omega$ from Eq. (32).
3: for each epoch : $i_{\text{ep}} \leftarrow 1, \ldots, N_{\text{ep}}$ do
4:   for each environment: $i_{\text{env}} \leftarrow 1, \ldots, N_{\text{g}}$ do
5:     $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega}_{i_{\text{env}}}$
6:     $E \sim \mathcal{C}_E$
7:     $f_0 \sim \mathcal{C}_{f_0}$
8:     for each edge server: $e' \leftarrow 1, \ldots, E$ do
9:       $f_{e'} \sim \mathcal{C}_{\boldsymbol{f}_{\mathcal{E}'}}$
10:    end for
11:    for each step: $t \leftarrow 1, \ldots, T$ do
12:      $a_t \sim \pi_{\boldsymbol{\phi}}(\cdot|\boldsymbol{s}_t)$
13:      $\boldsymbol{s}_{t+1} \sim \mathcal{T}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, a_t)$
14:      $\mathcal{D} \leftarrow \mathcal{D} \cup \{\langle \boldsymbol{s}_t, a_t, r_{\boldsymbol{\omega}}(\boldsymbol{s}_t, a_t), \boldsymbol{s}_{t+1} \rangle\}$
15:    end for
16:    for each update round: $i_{\text{up}} \leftarrow 1, \ldots, N_{\text{up}}$ do
17:      Sample experiences from $\mathcal{D}$
18:      Compute $J_Q(\boldsymbol{\theta}_i)$ for $i \in \{1,2\}$, $J_\pi(\boldsymbol{\phi})$, and
         $J(\alpha_H)$ by Eq. (A4), Eq. (A8), and Eq. (A11).
19:      Update the parameters according to Eq. (A5), Eq.
         (A6), Eq. (A10) and Eq. (A12):
20:      $\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_i - \lambda_Q \hat{\nabla}_{\boldsymbol{\theta}_i} J_Q(\boldsymbol{\theta}_i)$ for $i \in \{1,2\}$
21:      $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} - \lambda_\pi \hat{\nabla}_{\boldsymbol{\phi}} J_\pi(\boldsymbol{\phi})$
22:      $\alpha_H \leftarrow \alpha_H - \lambda_\alpha \hat{\nabla}_{\alpha_H} J(\alpha_H)$
23:      $\bar{\boldsymbol{\theta}}_i \leftarrow \beta \boldsymbol{\theta}_i + (1-\beta)\bar{\boldsymbol{\theta}}_i$ for $i \in \{1,2\}$
24:    end for
25:  end for
26: end for
27: Output policy $\pi_{\boldsymbol{\phi}}$
```

function by starting from any function $Q: \mathcal{S} \times \mathcal{A} \to \mathbb{R}^2$ and repeatedly applying the modified Bellman backup operator T^π which satisfies

$$T^\pi Q(\boldsymbol{s}_t, a_t) = r(\boldsymbol{s}_t, a_t) + \gamma \mathbb{E}_{\boldsymbol{s}_{t+1} \sim \rho_\pi}(V(\boldsymbol{s}_{t+1})), \tag{A2}$$

where $V(\cdot)$ is a soft state-value function of policy π, and it satisfies

$$V(\boldsymbol{s}_t) = \mathbb{E}_{a_t \sim \pi}[Q(\boldsymbol{s}_t, a_t) - \alpha_H \log(\pi(a_t|\boldsymbol{s}_t))]. \tag{A3}$$

Then we train soft Q-function parameters $\boldsymbol{\theta}_i$ for $i \in \{1,2\}$ to minimize the soft Bellman residual. Soft Bellman residual $J_Q(\boldsymbol{\theta}_i)$ is given by Eq. (A4), where $\mathcal{D}$ is a replay buffer of past experiences, and $Q_{\boldsymbol{\theta}_i}(\cdot)$ is the soft Q-function with parameters $\boldsymbol{\theta}_i$. Soft state-value $V_{\bar{\boldsymbol{\theta}}_i}(\boldsymbol{s}_{t+1})$ is estimated by a target Q-function network according to Eq. (A3). Based on $J_Q(\boldsymbol{\theta}_i)$, we update local soft Q-function parameters $\boldsymbol{\theta}_i$ by

$$\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_i - \lambda_Q \hat{\nabla}_{\boldsymbol{\theta}_i} J_Q(\boldsymbol{\theta}_i), \tag{A5}$$

where λ_Q is the learning rate of soft Q-function, and $\hat{\nabla}_{\boldsymbol{\theta}_i} J_Q(\boldsymbol{\theta}_i)$ is the approximated gradient of $J_Q(\boldsymbol{\theta}_i)$. Next, we update target soft Q-function parameters $\bar{\boldsymbol{\theta}}_i$ by

$$\bar{\boldsymbol{\theta}}_i \leftarrow \beta \boldsymbol{\theta}_i + (1-\beta)\bar{\boldsymbol{\theta}}_i, \tag{A6}$$

where β is a target smoothing coefficient. In the policy improvement step, we update policy π according to

$$\pi_{\text{new}} = \underset{\pi \in \Pi'}{\operatorname{argmin}} D_{\text{KL}} \left(\pi(\cdot \mid \boldsymbol{s}_t) \,\middle\|\, \frac{\exp\left(\frac{1}{\alpha_H} Q^{\pi_{\text{old}}}(\boldsymbol{s}_t, \cdot)\right)}{Z^{\pi_{\text{old}}}(\boldsymbol{s}_t)} \right) \tag{A7}$$

where $D_{\text{KL}}(\cdot)$ is the Kullback-Leibler (KL)-divergence function, and Π' is a policy search space that is applied to restrict the policy. The partition function $Z^{\pi_{\text{old}}}(\cdot)$ normalizes the policy distribution, ensuring that it sums up to a probability of 1 over the entire action space. We optimize policy parameters $\boldsymbol{\phi}$ to minimize the KL-divergence by the policy objective $J_\pi(\boldsymbol{\phi})$ which is given by Eq. (A8), where $Q_{\boldsymbol{\theta}_1}(\cdot, \boldsymbol{s}_t)$ and $Q_{\boldsymbol{\theta}_2}(\cdot, \boldsymbol{s}_t)$ are the Q-value vectors for all actions at state $\boldsymbol{s}_t$, with parameters $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$.

We denote the policy gradient direction for the reward of delay r_{T} as $\hat{\nabla}_{\boldsymbol{\phi}} J_{\pi,\text{T}}(\boldsymbol{\phi})$, and denote the policy gradient direction for the reward of energy consumption r_{E} as $\hat{\nabla}_{\boldsymbol{\phi}} J_{\pi,\text{E}}(\boldsymbol{\phi})$. The policy gradient direction for reward $r_{\boldsymbol{\omega}}$ is

$$\hat{\nabla}_{\boldsymbol{\phi}} J_{\pi,\boldsymbol{\omega}}(\boldsymbol{\phi}) = \boldsymbol{\omega}^T \times (\hat{\nabla}_{\boldsymbol{\phi}} J_{\pi,\text{T}}(\boldsymbol{\phi}), \hat{\nabla}_{\boldsymbol{\phi}} J_{\pi,\text{E}}(\boldsymbol{\phi})). \tag{A9}$$

Given the gradient directions of the delay objective and the energy consumption objective, a policy can reach the Pareto front by following a direction in ascent simplex [18]. An ascent simplex is defined by the convex combination of single–objective gradients.

Synthesizing the above, we update policy parameters $\boldsymbol{\phi}$ by

$$\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} - \lambda_\pi \hat{\nabla}_{\boldsymbol{\phi}} J_\pi(\boldsymbol{\phi}), \tag{A10}$$

where $\lambda_{\boldsymbol{\phi}}$ is the learning rate of policy parameters $\boldsymbol{\phi}$, and $\hat{\nabla}_{\boldsymbol{\phi}} J_\pi(\boldsymbol{\phi})$ is the approximated gradient of $J_\pi(\boldsymbol{\phi})$.

Finally, the temperature parameter α_H is learnable. The temperature objective is

$$J(\alpha_H) = \pi_t(\boldsymbol{s}_t)^T \left[-\alpha_H \left(\log\left(\pi_{\boldsymbol{\phi}(\boldsymbol{s}_t)}\right) + \bar{\mathcal{H}}\right)\right], \tag{A11}$$

where $\bar{\mathcal{H}}$ is a constant vector equal to the hyperparameter representing the target entropy. We update α_H by

$$\alpha_H \leftarrow \alpha_H - \lambda_\alpha \hat{\nabla}_{\alpha_H} J(\alpha_H), \tag{A12}$$

where λ_α is the learning rate of temperature parameter α_H, and $\hat{\nabla}_{\alpha_H} J(\alpha_H)$ is the approximated gradient of $J(\alpha_H)$.

References

[1] G. Cui, X. Li, L. Xu, and W. Wang, "Latency and energy optimization for mec enhanced sat-iot networks," *IEEE Access*, vol. 8, pp. 55 915–55 926, 2020.

[2] L. Lei, H. Xu, X. Xiong, K. Zheng, W. Xiang, and X. Wang, "Multiuser resource control with deep reinforcement learning in iot edge computing," *IEEE Internet of Things J.*, vol. 6, no. 6, pp. 10 119–10 133, 2019.

[3] J. Li, H. Gao, T. Lv, and Y. Lu, "Deep reinforcement learning based computation offloading and resource allocation for mec," in *2018 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2018, pp. 1–6.

$$J_Q(\boldsymbol{\theta}_i) = \mathbb{E}_{(\boldsymbol{s}_t, a_t)\sim\mathcal{D}}\left[\frac{1}{2}\left(Q_{\boldsymbol{\theta}_i}\left(\boldsymbol{s}_t, a_t\right) - \left(r\left(\boldsymbol{s}_t, a_t\right) + \gamma\mathbb{E}_{\boldsymbol{s}_{t+1}\sim\mathcal{T}}\left[V_{\bar{\boldsymbol{\theta}}_i}\left(\boldsymbol{s}_{t+1}\right)\right]\right)\right)^2\right], \forall i \in \{1, 2\} \tag{A4}$$

$$J_\pi(\boldsymbol{\phi}) = \mathbb{E}_{\boldsymbol{s}_t\sim\mathcal{D}}\left[\pi_t\left(\cdot, \boldsymbol{s}_t\right)^T\left[\alpha_H \log\left(\pi_{\boldsymbol{\phi}}\left(\cdot, \boldsymbol{s}_t\right)\right) - \min(Q_{\boldsymbol{\theta}_1}(\boldsymbol{s}_t, \cdot), Q_{\boldsymbol{\theta}_2}(\boldsymbol{s}_t, \cdot))\right]\right] \tag{A8}$$

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

[5] D. C. Nguyen, P. N. Pathirana, M. Ding, and A. Seneviratne, "Deep reinforcement learning for collaborative offloading in heterogeneous edge networks," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2021, pp. 297–303.

[6] F. Jiang, L. Dong, K. Wang, K. Yang, and C. Pan, "Distributed resource scheduling for large-scale mec systems: A multiagent ensemble deep reinforcement learning with imitation acceleration," *IEEE Internet of Things Journal*, vol. 9, no. 9, pp. 6597–6610, 2021.

[7] Y. Li, J. Li, Z. Lv, H. Li, Y. Wang, and Z. Xu, "Gasto: A fast adaptive graph learning framework for edge computing empowered task offloading," *IEEE Transactions on Network and Service Management*, 2023.

[8] T. Ren, J. Niu, and Y. Qiu, "Enhancing generalization of computation offloading policies in novel mobile edge computing environments by exploiting experience utility," *Journal of Systems Architecture*, vol. 125, p. 102444, 2022.

[9] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, "Fast adaptive task offloading in edge computing based on meta reinforcement learning," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 242–253, 2020.

[10] T. Wu, W. Jing, X. Wen, Z. Lu, and S. Zhao, "A scalable computation offloading scheme for mec based on graph neural networks," in *2021 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2021, pp. 1–6.

[11] Z. Hu, J. Niu, T. Ren, and M. Guizani, "Achieving fast environment adaptation of drl-based computation offloading in mobile edge computing," *IEEE Transactions on Mobile Computing*, 2023.

[12] F. Jiang, K. Wang, L. Dong, C. Pan, and K. Yang, "Stacked autoencoder-based deep reinforcement learning for online resource scheduling in large-scale mec networks," *IEEE Internet of Things J.*, vol. 7, no. 10, pp. 9278–9290, 2020.

[13] N. Yang, J. Wen, M. Zhang, and M. Tang, "Multi-objective deep reinforcement learning for mobile edge computing," in *2023 21st international symposium on modeling and optimization in mobile, ad hoc, and wireless networks (WiOpt)*. IEEE, 2023, pp. 1–8.

[14] J. Chang, J. Wang, B. Li, Y. Zhao, and D. Li, "Attention-based deep reinforcement learning for edge user allocation," *IEEE Transactions on Network and Service Management*, 2023.

[15] Z. Gao, L. Yang, and Y. Dai, "Fast adaptive task offloading and resource allocation in large-scale mec systems via multi-agent graph reinforcement learning," *IEEE Internet of Things Journal*, 2023.

[16] "Ieee standard for telecommunications and information exchange between systems - lan/man specific requirements - part 11: Wireless medium access control (mac) and physical layer (phy) specifications: High speed physical layer in the 5 ghz band," *IEEE Std 802.11a-1999*, pp. 1–102, 1999.

[17] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," in *Proceedings of the 19th international conference on World wide web*, 2010, pp. 661–670.

[18] S. Parisi, M. Pirotta, N. Smacchia, L. Bascetta, and M. Restelli, "Policy gradient approaches for multi-objective sequential decision making," in *2014 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2014, pp. 2323–2330.