Joint Caching, Communication, Computation Resource Management in Mobile-Edge Computing Networks

Mingqi Han*, Xinghua Sun*, Xijun Wang†, Wen Zhan*, Xiang Chen†
*School of Electronics and Communication Engineering, Shenzhen Campus of Sun-Yat sen University, Shenzhen, China
†School of Electronics and Information Technology, Sun-Yat sen University, Guangzhou, China
Email:{hanmq}@mail2.sysu.edu.cn, {sunxinghua, wangxijun, zhanwen6, chenxiang}@mail.sysu.edu.cn

***Abstract*—Mobile-edge Computing (MEC) has now emerged as a complement to cloud computing, providing computational capacity for the resources-constrained edge devices. Recently, intelligent computation offloading and cache placement stands as effective approaches to enhance the performance of dynamic MEC networks. In this paper, we propose an online centralized joint resource management approach, named Transformer-based Actor-Critic (TAC), to minimize the task execution time subject to resource constraints. We decouple this mixed-integer non-linear programming (MINLP) problem into a non-convex offloading decision part and a convex joint resources allocation part, and propose the TAC approach to address the non-convex task offloading problem with low computational complexity. In the joint resources management problem, the high-dimensional state-action space is addressed by the transformer-based actor-critic architecture. Through the proposed TAC, the joint cache, communication and computation resource management can be obtained without the knowledge of future task arrivals. Simulation results demonstrate that the TAC can save 48.4% average task execution time with only 2.3% additional computation delay compared to *Random* with lowest computational complexity. In particular, it further demonstrates great generalization ability to enhance the performance in untrained scenarios.**

***Index Terms*—Mobile Edge Computing, Deep Reinforcement Learning, Joint Resource Management**

I. Introduction

With the rapid development of Internet-of-Things (IoTs) devices, it is anticipated that there will be 50 billion IoTs devices in the whole network, which enables wireless user equipments (UEs) evolve into both content producers and consumers [1]. Those large-scale interconnected user devices will generate diverse data traffic and data requirements, including caching, communication and computation.

Since most of UEs in the edge network only have limited computational capacity, mobile-edge computing (MEC) plays a crucial role in providing the computational capacity [2]. With edge servers deployed in MEC network, both caching services and computing capabilities can be provided to UEs to save computation energy and time cost. However, offloading all tasks to the edge servers can significantly degrade the computation performance due to signalling overhead and queueing delay. To address it, intelligent computation offloading and cache placement is proposed by selectively offloading and caching suitable tasks to the edge servers [3]–[5]. Through efficient offloading and caching policies, both the time and energy consumption for task computation and communication can be minimized. In [4], an offline Deep Learning (DL) cache placement scheme was proposed to save roughly 33% energy consumption compared to the greedy caching scheme. In [5], a Lyapunov-guided Deep Reinforcement Learning (DRL) approach was proposed to transfer the multi-stage optimization problem into conventional problem, and apply the model-free DRL approach to decide the offloading decision.

Considering multiple edge servers in the MEC network sharing computation and communication resources, an efficient joint resource allocation is also crucial to meet the demand of large-scale tasks computation [3], [4], [6]. When each edge server operates independently, it may struggle to consistently handle the computation and cache demands from a big-data stream, rendering the necessity to dynamically optimize resources allocation. In [6], the joint resource allocation problem was regarded as the proximal upper bound problem, and the block successive upper bound minimization was applied to obtain better joint resource allocation. In [3], an closed-form expression of optimal transmission power and CPU frequency was derived given the specific offloading decision.

In this paper, we consider a more general MEC network involving multi-level devices, including multiple UEs, edge servers and the cloud center. In the considered scenario, since the high-level devices commonly cover multiple low-level devices, not only the number of UE at the lowest level increases significantly, but the destination of task offloading also increases linearly with respect to the number of levels, resulting in a high-dimensional state-action space. In such scenarios, conventional DRL approaches struggle to handle the complex problem of multiple interrelated actions. To address it, we propose the Transformer-based Actor Critic (TAC) by combing the transformer and the actor-critic architecture [7]. Utilizing the transformer, the proposed TAC algorithm can efficiently address the high-dimensional action problem, and enhance the evaluation ability of critic on high-dimensional state space, resulting in significantly better performance.

The rest of paper is organized as follows. In Sec. II, the

This work of Xinghua Sun was supported in part by the National Key Research and Development Program of China under Grant 2023YFB2904100, and in part by Shenzhen Science and Technology Program (Grant No. ZDSYS20210623091807023). The work of Wen Zhan was supported in part by Shenzhen Science and Technology Program (No.RCBS20210706092408010), and in part by National Natural Science Foundation of China under Grant 62001524.

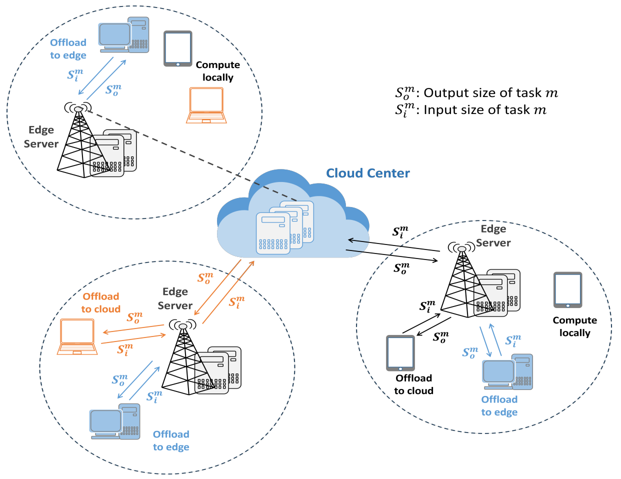

Fig. 1. The considered multi-user MEC network with single cloud center, multiple edge servers and different types of UEs.

system model and the problem of interest are described. In Sec. A., the decoupled optimization problem is formulated, and the proposed TAC algorithm is illustrated. In Sec. IV, the performance of TAC algorithm is evaluated. Finally, the paper is concluded in Sec. V.

II. System Model and Problem of Interest

A. System Model

As shown in Fig. 1, we consider an MEC network composed of a single cloud center, a set of edge servers $\mathcal{K}$ and a set of UEs $\mathcal{N}$. Each edge server is deployed with a Base Station (BS) and covers multiple UEs. For simplicity, we assume that UEs can only communicate with their associated edge server, and the set of UEs associated with edge server $k \in \mathcal{K}$ is denoted as $\mathcal{N}_k \subset \mathcal{N}$. The set of tasks is denoted as $\mathcal{M}$, and we adopt a common computation model to characterize them [8]. Each computation task $m \in \mathcal{M}$ is represented by the vector (S_i^m, C^m, S_o^m), in which S_i^m and S_o^m denote the size of input data and the output computed data of task m, respectively, and C^m denotes the required CPU cycle per bit of input data.

In order to ensure successful joint resource allocation and decision-making for offloading and caching in the MEC network, we divide time into rounds as illustrated in Fig. 2. Each round begins with each UE being assigned a single task randomly selected from the set of tasks $\mathcal{M}$. Notably, different types of UEs have their own independent task arrival distributions, resulting in varying expected numbers of each type of tasks arrived at different types of UEs. After receiving tasks, UEs make task offloading decision, i.e., compute locally, upload to the edge server or further upload to cloud center. If the task is offloaded, the edge server or cloud center will compute the task and send the result of task back to the UE. Each round ends only when all tasks have been computed, and the duration time of round t is denoted T_t, which equals to the maximum task execution time among all UEs.

B. Problem Formulation

In the considered network, the allocation on the communication and computation resources of edge servers, the caching decision of edge servers, and the task offloading decision of UEs are under the control of cloud center. For computation resource, the allocation ratio among edge servers at each round t are denoted as $\{a_t^k | k \in \mathcal{K}\}$, subject to the resource constraint $\sum_{k\in\mathcal{K}} a_t^k = 1$. Since the computation resource linearly relates to the CPU frequency [8], we consider the computation resource as the CPU frequency (in CPU rounds per second) in the following. Subsequently, the time-varying computation capacity of each edge server can be represented as $a_t^k F_E$, in which F_E represents the total computation resource for edge servers. While the computation capacity of cloud center and UEs are fixed over time, which are denoted as F_C and F_{UE}, respectively.

For communication resource, the allocation ratios among edge servers at each round t are denoted as $\{b_t^k | k \in \mathcal{K}\}$, subject to $\sum_{k\in\mathcal{K}} b_t^k = 1$. Subsequently, the available bandwidth of the transmission between edge server k and its associated UEs $\mathcal{N}_k$ is given by $b_t^k B$, in which B represents the total bandwidth of the MEC network. Each UE associated with the same edge server communicate using the Time-Division Multiple Access (TDMA) manner. The transmission between cloud center and edge server k uses the same available bandwidth $b_t^k B$.

For the cache resources, we assume that only the edge servers have the capability to store the results of offloaded tasks. By caching the results of tasks, the edge servers can directly provide the cached results to corresponding UEs, eliminating the need for redundant computations. The storage capacity (in bits) of each edge server k is denoted as S^k. The edge server can cache the result of offloaded task m only when the current cache capacity is sufficient.

Since the UEs have only limited computation resource, offloading the tasks to the associated edge server or the cloud center can significant accelerate the tasks computing time. The offload decision of UE n at the round t is given by

$$x_t^n = \begin{cases} 2, & \text{if task is offloaded to the cloud center} , \\ 1, & \text{if task is offloaded to nearest edge server} , \\ 0, & \text{otherwise} \end{cases} . \quad (1)$$

When $x_t^n = 2$, since UEs $n \in \mathcal{N}_k$ can only communicate with its edge server k, the task will be transmitted from the edge server first, and then to the cloud center. Subsequently,

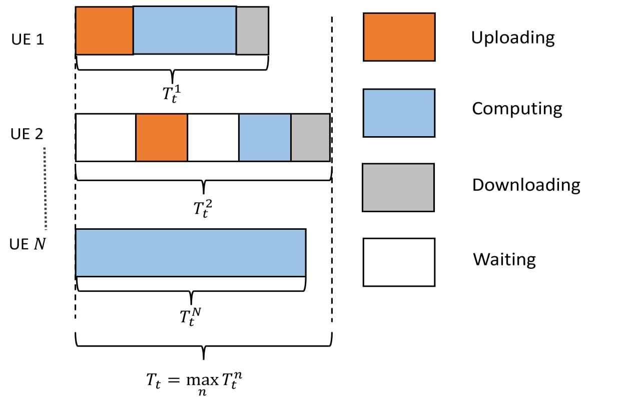

Fig. 2. Illustration of the workflow for the MEC system. The duration of each decision time interval equals to the maximum computing time among UEs.

the edge server can cache the result of offloaded task when its current cache capacity is sufficient and the task has been offloaded, i.e., $x_t^n \in \{1,2\}$. The cache decision of edge server k for each offloaded task m at the round t can be given by

$$c_t^k(m) = \begin{cases} 1, \text{ cache the result of offloaded task } m, \\ 0, \text{ do not cache the result of task } m, \end{cases} . \tag{2}$$

according to (1). We denote the cached results of tasks in edge server k as $C_t^k(m) \in \{0,1\}$, where $C_t^k(m) = 0$ represent task m has not cached in edge server k at round t. Subsequently, the constraint of cache capacity is given by $S_k \geq \sum_m C_t^k(m) S_o^m$. Then, we can derive the task computation time $\mathcal{T}_{1,t} = \{T_{1,t}^n | n \in \mathcal{N}\}$ according to the offloading decision and cached status, i.e.,

$$T_{1,t}^n = \begin{cases} \frac{S_i^m C^m}{F_{UE}}, \text{ if } x_t^n = 0, \\ (1 - C_t^k(m)) \frac{S_i^m C^m}{a_t^k F_E}, \text{ if } x_t^n = 1, \\ \frac{S_i^m C^m}{F_C}, \text{ if } x_t^n = 2, \end{cases} . \tag{3}$$

To derive the task communication time, we first obtain the instantaneous data rates between UEs, edge servers and the cloud center, which are given by

$$\begin{aligned} R(n,k) &= b_t^k B \log(1 + \frac{P|h(k,n)|^2}{\sigma^2}) \\ R(c,k) &= b_t^k B \log(1 + \frac{P|h(c,k)|^2}{\sigma^2}) \end{aligned}, \tag{4}$$

where $P \in \{P_{UE}, P_E, P_C\}$ denotes the fixed transmission power of device including UEs, edge servers and the cloud center, respectively. $h(k,n)$ denotes the channel gain between UE n and associated edge server k, and $h(c,k)$ denotes the channel gain between the cloud center and edge server k. For simplicity, we assume each device has the same power of the Gaussian noise σ^2. Then, the task communication time $\mathcal{T}_{2,t} = \{T_{2,t}^n | n \in \mathcal{N}\}$ can be derived as

$$T_{2,t}^n = \begin{cases} 0, \text{ if } x_t^n = 0, \\ \frac{S_i^m}{R(n,k)} + \frac{S_o^m}{R(k,n)}, \text{ if } x_t^n = 1, \\ \frac{S_i^m}{R(k,c)} + \frac{S_o^m}{R(c,k)}, \text{ if } x_t^n = 2, \end{cases} . \tag{5}$$

Then, since the task offloading and computation procedure is sequentially executed over time, we denote the queueing matrix for computation on edge servers as $\mathcal{Q}_{11,t} = \{q_{11,t}(n_1,n_2) | n_1, n_2 \in \mathcal{N}\}$ and on the cloud center as $\mathcal{Q}_{12,t} = \{q_{12,t}(n_1,n_2) | n_1, n_2 \in \mathcal{N}\}$, where

$$\begin{aligned} q_{11,t}(n_1,n_2) &= \begin{cases} 1, \text{ if } x_t^{n_1} = x_t^{n_2} = 1 \text{ and } n_1, n_2 \in \mathcal{N}_k, \\ 1, \text{ if } n_1 = n_2, \\ 0, \text{ otherwise }, \end{cases} \\ q_{12,t}(n_1,n_2) &= \begin{cases} 1, \text{ if } x_t^{n_1} = 2 \text{ and } x_t^{n_2} = 2, \\ 1, \text{ if } n_1 = n_2, \\ 0, \text{ otherwise }, \end{cases} \end{aligned}, \tag{6}$$

and denote the queueing matrix for communication as $\mathcal{Q}_{2,t} = \{q_{2,t}(n_1,n_2) | n_1, n_2 \in \mathcal{N}\}$, where

$$q_{2,t}(n_1,n_2) = \begin{cases} 1, \text{ if } x_t^{n_1} = x_t^{n_2} = 1 \text{ and } n_1, n_2 \in \mathcal{N}_k, \\ 1, \text{ if } x_t^{n_1} = 2 \text{ and } x_t^{n_2} = 2, \\ 0, \text{ otherwise }, \end{cases} . \tag{7}$$

$\mathcal{Q}_{11,t}, \mathcal{Q}_{12,t}$ and $\mathcal{Q}_{2,t}$ represent that UE n_1 and n_2 needs to wait for each other to complete task computing and communication in an TDMA manner. Finally, we can obtain the duration time T_t of each round t as

$$\begin{aligned} T_t =& \| \mathcal{T}_{1,t}(0) + \mathcal{Q}_{11,t} \cdot \mathcal{T}_{1,t}(1) \\ &+ \mathcal{Q}_{12,t} \cdot \mathcal{T}_{1,t}(2) + \mathcal{Q}_{2,t} \cdot \mathcal{T}_{2,t}(1) + \mathcal{T}_{2,t}(2) \|_\infty \end{aligned}, \tag{8}$$

where $\mathcal{T}_t(x) = \{T_{1,t}^n \text{ if } x_t^n = x \text{ else } 0 | n \in \mathcal{N}\}$ is the part of matrix $\mathcal{T}_t$ comprising by the elements of specific offloading decision x. In this paper, we aim to proposed an online algorithm to enhance the performance of the MEC network by minimizing the task execution time, with the cache, computation and communication resources constraints. By denoting $\mathbf{x_t} = \{x_t^1, \ldots, x_t^N\}$, $\mathbf{c_t} = \{c_t^1, \ldots, c_t^K\}$, $\mathbf{a_t} = \{a_t^1, \ldots, a_t^K\}$ and $\mathbf{b_t} = \{b_t^1, \ldots, b_t^K\}$, and letting $\mathbf{x} = \{\mathbf{x_t}\}_t^T$, $\mathbf{c} = \{\mathbf{c_t}\}_t^T$, $\mathbf{a} = \{\mathbf{a_t}\}_t^T$ and $\mathbf{b} = \{\mathbf{b_t}\}_t^T$, the optimization problem can be formulated as:

$$\begin{aligned} & \underset{\mathbf{x,c,a,b}}{\text{minimize}} \lim_{T \to \infty} \frac{1}{T} \cdot \sum_{t=1}^{T} T_t \\ & \text{subject to:} \\ & a_t^k \in [0,1], \; b_t^k \in [0,1], \sum_{k=1}^{K} b_t^k \leq 1, \; \sum_{k=1}^{K} a_t^k \leq 1, \; \forall k, t \\ & x_t^k \in \{0,1,2\}, \; c_t^k(m) \in \{0,1\}, \; S_k \geq \sum_m C_t^k(m) S_o^m, \; \forall m, k, t \end{aligned} . \tag{9}$$

III. TAC Algorithm

Given that the optimization problem in (9) involves the infinite norm in the objective function, i.e., T_t, along with the binary caching and ternary offloading decision constraints, the problem can be regarded as a Mixed-Integer Non-linear Programming (MINLP) problem. To address the MINLP problem, we partition the original problem (9) in each round t into two parts, the non-convex part and the convex part. First, we include the non-convex binary caching $\mathbf{c}$ and ternary offloading decision $\mathbf{x}$ constraints in one part:

$$\begin{aligned} & \underset{\mathbf{x_t,c_t}}{\text{minimize}} \; T_t^*(\mathbf{x_t}, \mathbf{c_t}) \\ & \text{subject to:} \\ & x_t^k \in \{0,1,2\}, \; c_t^k(m) \in \{0,1\}, \; S_k \geq \sum_m C_t^k(m) S_o^m, \; \forall m, k, t \end{aligned}, \tag{10}$$

in which $T_t^*(\mathbf{x_t}, \mathbf{c_t}) = \min_{\mathbf{a_t}, \mathbf{b_t}} T_t(\mathbf{x_t}, \mathbf{c_t}, \mathbf{a_t}, \mathbf{b_t})$ denotes the minimum value of T_t given the specific binary caching decision $\mathbf{c_t}$ and ternary offloading decision $\mathbf{x_t}$. After obtaining the offloading decision, the rest part of caching decision and resources allocation can be addressed as a conventional convex problem, which is given by,

$$\begin{aligned} & \underset{\mathbf{a_t,b_t}}{\text{minimize}} \; T_t(\mathbf{x_t}, \mathbf{c_t}, \mathbf{a_t}, \mathbf{b_t}) \\ & \text{subject to:} \\ & a_t^k \in [0,1], \; b_t^k \in [0,1], \sum_{k=1}^{K} b_t^k \leq 1, \; \sum_{k=1}^{K} a_t^k \leq 1, \; \forall k, t, \end{aligned}, \tag{11}$$

which can be directly solved by the conventional convex optimization approaches. For (10), exhaustive searching for the ternary offloading and binary cache decision has excessive computational complexity, which requires 3^N and 2^N iterative times, respectively. Given the computational complexity of convex optimization on searching the optimal joint resources allocation for (11) as $\mathcal{O}(N^{3.5})$, the overall computation complexity for joint resource management is given by $\mathcal{O}(N^{3.5} \cdot 3^N)$, which may cause unaffordable exponential increasing computational complexity and delay with respect to number of UEs N.

In practical dynamic scenarios, such MINLP problem must be solved repeatedly when new tasks arrive. Consequently, employing conventional optimization algorithms in such MEC environment becomes unaffordable. Moreover, the computation delay introduced by the conventional optimization algorithms will further degrade the performance of the MEC network. To enhance its performance while reducing the computational complexity, we propose the TAC algorithm and decouple the offloading decision and caching decision. Only when a specific offloading decision is given, the caching policy can be derived, leading us to primarily focus on optimizing the offloading decision. Since the future task arrivals are not available, we introduce the Partial Observation Markov decision process (POMDP) model formulation of the proposed TAC, i.e., state, action and reward design. Then, we will illustrate the TAC algorithm, along with the transformer architecture employed to tackle the high-dimensional state and action space.

A. DRL Model Formulation

In this section, we regard the cloud center as agent and the offloading decision as the POMDP since the future tasks arrival status is unobservable for the cloud center. During each round, the agent first observes the environmental state, makes an efficient offloading decision according to state, and then receives a reward.

A.1 State-Action

As aforementioned, the transformer has the feature that the number of output is the same as input utilizing the encoder-decoder architecture. By introducing the transformer to address the high-dimensional state and action space, we define the state of each UE and aggregate them to form the agent's state as input of TAC model. The state of UE n is given by

$$s_t^n = \{T_{1,t}^{*,n}(0), T_{2,t}^{*,n}(0), T_{1,t}^{*,n}(1), T_{2,t}^{*,n}(1), T_{1,t}^{*,n}(2), T_{2,t}^{*,n}(2)\}, \tag{12}$$

in which $T_{1,t}^{*,n}(x), T_{2,t}^{*,n}(x)$ denotes the value of $T_{1,t}^n$ and $T_{2,t}^n$ of (3) and (5) given by $x_t^n = x, a_t^k = b_t^k = 1$, respectively. Then, the state of TAC agent s_t is given by

$$s_t = \{s_t^n\}_{n=1}^N \in \mathbb{R}^{N\times 6}. \tag{13}$$

For the agent's action, since we regard the offloading decision as the POMDP, the action is directly the stack of the ternary offloading decision of all UEs $\mathbf{x_t}$.

A.2 Reward

As our goal is to minimize the duration time T_t in each round t with the joint cache, communication and computation resource constraint, we take the negative value of it in each round t as the reward, i.e.,

$$r_t = -\alpha T_t, \tag{14}$$

where α is the coefficient to limit the range of reward r_t and stabilizes the convergence of TAC model. Through this reward design, the optimization problem can be addressed when the proposed TAC model converged.

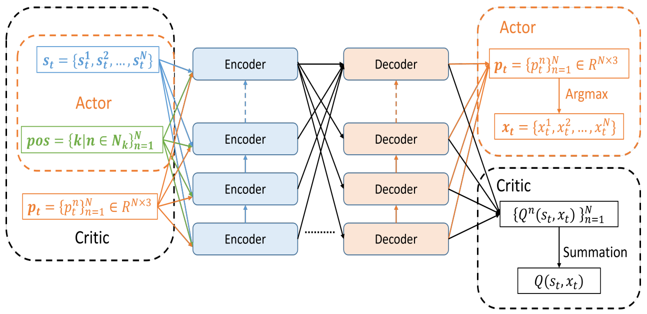
Actor
$s_t = \{s_t^1, s_t^2, \ldots, s_t^N\}$
$pos = \{k|n \in N_k\}_{n=1}^N$
$p_t = \{p_t^n\}_{n=1}^N \in R^{N\times 3}$
Critic
Encoder
Encoder
Encoder
Encoder
Decoder
Decoder
Decoder
Decoder
Actor
$p_t = \{p_t^n\}_{n=1}^N \in R^{N\times 3}$
Argmax
$x_t = \{x_t^1, x_t^2, \ldots, x_t^N\}$
Critic
$\{Q^n(s_t, x_t)\}_{n=1}^N$
Summation
$Q(s_t, x_t)$

Fig. 3. Architecture of the transformer in the TAC. Multiple encoders take the tasks of all UEs as input, the associated id of edge server as position embedding, and pass them through the decoder to output the same number of offloading decisions of each UE.

B. Transformer Architecture

As aforementioned, we introduce the transformer architecture in the proposed TAC algorithm instead of widely-used multi-layer perceptron (MLP) in DRL approaches, which has two-fold reason. First, conventional MLP cannot efficiently tackle the high-dimensional state space $S \in \mathbb{R}^{N\times 6}$ and action space $X \in \mathbb{R}^{N\times 3}$ due to dimensional disaster and low sample efficiency. Second, using MLP requires to flatten the input vector, which will destroy the inherent representation between the state of each UE and degrade the performance.

To address these challenges, we introduce the transformer architecture, and extend the position embedding into the considered MEC network. As illustrated in Fig. 3, each individual encoder takes the same state s_t and position embedding pos as input. In particular, since the UEs associated the same edge server exhibit coupling computation and communication time when $x_t^n \in \{1,2\}$, we use the id of associated edge server k of each UE $n \in \mathcal{N}_k$ as the position embedding to capture the intrinsic relationships among UEs. Then, the output of encoders are pass to the decoders, and then output the probability $P_t = \{P_t^n\}_{n\in\mathcal{N}}$ of each offloading decision $x_t^n \in \{0,1,2\}$ for all UEs. Finally, those offloading decisions with maximum probability are selected as action, i.e., $\mathbf{x_t} = \underset{x_t^n \in \{0,1,2\}}{\arg\max}\, P_t$.

C. Algorithm Overview

The overall procedure of the proposed TAC is illustrated in Fig. 4. During each round, the cloud center collects the tasks status of UEs s_t, then obtain the offloading decision x_t and broadcast to each UE to perform. Finally, the cloud center obtain the reward r_t from the environment and restore the tuple s_t, x_t, r_t into memory for the following training. In particular, we consider to pre-train the critic network using the random offloading decision in the proposed TAC. This random offloading decision can efficiently collect data for the offline training of critic network since it has low computational complexity to provide sufficient data for the training of transformers. We use the Mean Squared Error (MSE) as loss function of the critic, i.e.,

$$L_{critic} = \left(Q\left(s_t, p_t\right) - r_t + \gamma Q\left(s_{t+1}, p_{t+1}\right)\right)^2, \tag{15}$$

in which $\left(Q\left(s_t, p_t\right)\right)$ is the Q-value represent the expected cumulative reward of offloading decision probability p_t in

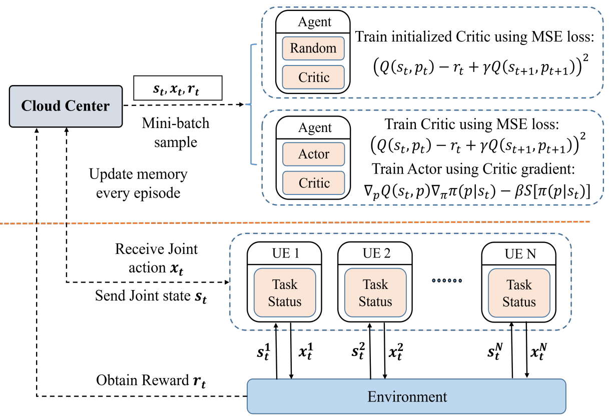
Agent
Random
Critic
Train initialized Critic using MSE loss:
$(Q(s_t,p_t) - r_t + \gamma Q(s_{t+1},p_{t+1}))^2$
s_t, x_t, r_t
Cloud Center
Mini-batch sample
Agent
Actor
Critic
Train Critic using MSE loss:
$(Q(s_t,p_t) - r_t + \gamma Q(s_{t+1},p_{t+1}))^2$
Train Actor using Critic gradient:
$\nabla_p Q(s_t,p)\nabla_\pi \pi(p|s_t) - \beta S[\pi(p|s_t)]$
Update memory every episode
Receive Joint action x_t
Send Joint state s_t
UE 1
Task Status
UE 2
Task Status
UE N
Task Status
s_t^1 x_t^1 s_t^2 x_t^2 s_t^N x_t^N
Obtain Reward r_t
Environment

Fig. 4. Illustration of the TAC procedure. First, train the critic network using random offloading action. Then, using the pre-trained critic to accelerate the convergence of both actor and critic network.

specific state s_t. In particular, we introduce p_t instead of x_t to address the problem that the $\arg\max$ function is non-differentiable, which enables the gradient of critic can be pass to the actor for gradient descent. After pre-training the critic, it can evaluate the Q-value $Q(s_t, x_t)$ accurately and provide the approximate gradient of the non-differentiable function $T_t^*(\mathbf{x_t}, \mathbf{c_t})$. Then, the actor can utilize the gradient $\partial Q(s_t, p)/\partial x$ to efficiently optimize the offloading policy. The loss function of actor is given by,

$$L_{actor} = \nabla_p Q(s_t, p)|_{p=\pi(p|s_t)} \nabla_\pi \pi(p \mid s_t) - \beta S[\pi(p \mid s_t)], \tag{16}$$

where $\pi(p \mid s_t)$ denotes the actor network, and $\beta S[\pi(x \mid s_t)]$ represents the entropy-loss of the output probability p given state s_t as similar to that in the SAC algorithm. Through the introduction of entropy-loss, the proposed TAC algorithm can have better exploration ability and find better offloading actions.

IV. Simulation Results and Discussions

In this section, we evaluate the proposed TAC algorithm by comparing simulation results with other methods. In the following, we first introduce the simulation setup, and then the detailed performance evaluations will be presented under different scenarios.

A. *Simulation Setting*

In the following, we consider that the number of UEs associated with each edge server follows the uniform distribution, and their positions follow the two-dimensional uniform distribution taking the position of the edge server as center. Edge servers are uniformly distributed on a fixed-radius circumference around the cloud center. The detailed parameters of MEC network and training are presented in Table I and Table II, respectively.

To present a comprehensive comparison, we introduce the *Random* method, i.e., the probability of ternary offloading decision of each UE p_t^n is randomly selected, and extend the *Random* and TAC approaches using additional generated actions as baselines similar to that in [5]. In particular, the additional actions is generated by adding Gaussian noise with different powers on the decision probability p_t^n. Then, the values of $T_t^*(\mathbf{x_t}, \mathbf{c_t})$ of all generated offloading decisions $\mathbf{x_t}$ are calculated, and select the decision with minimum value of $T_t^*(\mathbf{x_t}, \mathbf{c_t})$ as the actual decision. Such approach requires much larger computational complexity, and is only introduced to evaluate the performance of proposed TAC algorithm.

TABLE I
MEC Network Parameters [1]

Parameters	Value
Radius of edge server	100 or 200 meters
Radius of Cloud Center	200 or 300 meters
Num of edge servers	6 or 8
Num of UEs associated with edge server	$\{3,4,5\}$
Num of tasks	$M = 25$
Num of types of UEs	6
Num of types of tasks	25
Total Bandwidth	80 MHz
Total Computation Resources	86.4 or 115.2 GHz
Transmission Power P_C, P_E, P_{UE}	$40, 30, 23.9$ dBm
Distribution of input data size	$[5, 20]\ Mbits$
Distribution of output data size	$[2, 40]\ Mbits$
Distribution of CPU rounds	$[600, 1200]$

TABLE II
Hyper-parameters of TAC

Parameters	Value	Parameters	Value
Learning rate	$2 \cdot 10^{-5}$	Num of rounds	600
Entropy coefficient β	0.01	Optimizer	AdamW

B. *Performance Metrics*

The following metrics are used to evaluate the performance of the proposed TAC algorithm.

- *Average execution time:* We demonstrate the average task execution time in each episode to evaluate the performance of the MEC network.
- *Computation delay:* To evaluate the computational complexity of the joint resource management approaches, we introduce the actual computation delay ratios of different approaches to the *Random* approach with lowest computational complexity in the same platform.

C. *Performance Comparison*

In this section, we consider the scenario comprising one cloud center and 6 edge servers, with the number of UEs associated with each edge server of edge servers ranging from 3 to 5, resulting in an expected number of UEs $N = 24$. The performance comparison is illustrated in Fig. 5, and the computation delay comparison is presented in Table III. First, the proposed TAC approach can achieve 48.4% task execution time reduction compared to the *Random* with only 2.3% additional computational delay. Then, the TAC can also significantly outperform the extended *Random* with 20 and 30 additional generated actions with $36.8\%, 33.7\%$ execution time reduction and $1744.1\%, 2398.4\%$ computation delay reduction, respectively. Furthermore, the TAC can achieve similar performance to itself with 20 additional generated actions, which also illustrates that the TAC can efficiently address this

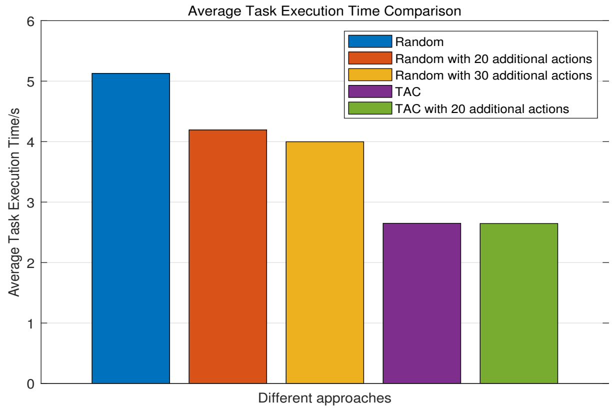

Fig. 5. Performance comparison between *Random* and proposed TAC algorithm with different number of generative actions. Notably, 20 and 30 represent that there are 20 and 30 additional actions are generated in each decision procedure, respectively.

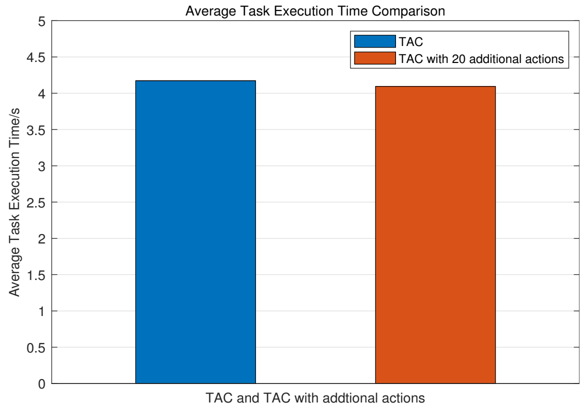

Fig. 6. Performance comparison between *Random* and proposed TAC algorithm with different number of generative actions. In particular, the TAC model is completely trained in the different scenario.

MINLP problem without need of additional generated actions.

TABLE III
THE COMPUTATION DELAY COMPARISON

Method	Computation Delay Ratio
Random	100%
Random with 20 additional actions	1846.4%
Random with 30 additional actions	2800.7%
TAC	102.3%
TAC with 20 additional actions	1857.9%

In particular, comparing the computation delay of extended *Random* to that of *Random*, the additional generated actions, i.e., 20 and 30, lead to computation delays that are approximately proportional to the number of additional action, i.e., 1846.4% and 2800.7%. Moreover, the extended TAC has similar computation delay as that of extended Random, indicating that the complexity of convex optimization in computing the optimal resource allocation (100%) is much higher than that of TAC model for determining offloading action (2.3%). Therefore, obtaining offloading action with need of additional generated actions is significant to reduce computation delay.

D. Generalization Ability

In this section, we consider to evaluate the generalization ability of the proposed TAC by using the test scenario different from training. The TAC model is the same as that in Fig. 5, and the test scenario includes 8 edge servers and expected number of UEs $N = 32$, in which the radius of edge servers and cloud center becomes 200 and 300, respectively. As presented in Figure 6, the performance of TAC degrades in untrained scenarios, i.e., it is slightly worse than itself with 20 additional actions, approximately 1.9%, which indicates that the proposed TAC can also exhibit great generalization ability in more challenging untrained scenarios.

V. CONCLUSION AND FUTURE WORK

In this paper, we consider a more general multi-level MEC network, including a cloud center, multiple edge servers and UEs. Aiming at minimizing the maximum task execution time in each round, we formulate a non-convex MINLP optimization problem, and decouple it into the convex part and non-convex part. To reduce the computational complexity and tackle the high-dimensional state-action space of the non-convex ternary offloading decision, we propose the TAC by combing transformer and Actor-Critic. Simulation results illustrate that the proposed TAC algorithm can save 48.4% task execution time, with only 2.3% additional computational computation delay compared to the *Random* with lowest computational complexity. Moreover, the TAC algorithm shows great generalization ability to maintain considerable performance even in untrained scenarios.

REFERENCES

[1] H. Jin, L. Su, D. Chen, K. Nahrstedt, and J. Xu, “Quality of information aware incentive mechanisms for mobile crowd sensing systems,” in *Proceedings of the 16th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2015, pp. 167–176.

[2] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, “A survey on mobile edge computing: The communication perspective,” *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[3] J. Yan, S. Bi, Y. J. Zhang, and M. Tao, “Optimal task offloading and resource allocation in mobile-edge computing with inter-user task dependency,” *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 235–250, 2020.

[4] J. Chen, H. Xing, X. Lin, A. Nallanathan, and S. Bi, “Joint resource allocation and cache placement for location-aware multi-user mobile-edge computing,” *IEEE Internet of Things Journal*, vol. 9, no. 24, pp. 25 698–25 714, 2022.

[5] 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.

[6] A. Ndikumana, N. H. Tran, T. M. Ho, Z. Han, W. Saad, D. Niyato, and C. S. Hong, “Joint communication, computation, caching, and control in big data multi-access edge computing,” *IEEE Transactions on Mobile Computing*, vol. 19, no. 6, pp. 1359–1374, 2020.

[7] K. Han, A. Xiao, E. Wu, J. Guo, C. Xu, and Y. Wang, “Transformer in transformer,” *Advances in Neural Information Processing Systems*, vol. 34, pp. 15 908–15 919, 2021.

[8] H. Xing, J. Cui, Y. Deng, and A. Nallanathan, “Energy-efficient proactive caching for fog computing with correlated task arrivals,” in *2019 IEEE 20th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, 2019, pp. 1–5.