Collaborative Service Caching, Task Offloading, and Resource Allocation in Caching-Assisted Mobile Edge Computing

Chaogang Tang, *Member, IEEE*, Yao Ding, Shuo Xiao, Zhenzhen Huang, and Huaming Wu, *Senior Member, IEEE*

***Abstract*—Mobile Edge Computing (MEC) revolutionizes the traditional cloud-based computing paradigm by moving resources in proximity to the network edge, aiming to cater to the rigorous requirements of emerging latency-sensitive applications. However, the escalating resource demands intensify the competition among user devices (UDs). Thus, it is essential to coordinate task offloading and resource scheduling while ensuring fairness among users in MEC. Despite the crucial role of user fairness in motivating task offloading in MEC, it is often overlooked in existing literature. Therefore, we in this paper propose a caching-enhanced MEC framework and formulate a collaborative service caching, task offloading, and multi-resource allocation problem to maximize average user satisfaction. Multiple factors contribute to the difficulty in solving the optimization problem, including constrained resource capabilities, user mobility, service heterogeneity, and spatial demand coupling. Consequently, we transform the origin problem into two distinct subproblems – the service caching and task offloading problem, and the multi-resource allocation problem, respectively. Then, the Advantage Actor-Critic (A2C) based approach is proposed to address the former problem, while a Lagrangian duality-based approach is adopted to tackle the latter problem. The simulation results demonstrate the superior performance of the proposed solution in comparison to several baseline methods.**

***Index Terms*—Service caching, task scheduling, multi-resource allocation, advantage actor-critic, edge computing.**

I. Introduction

MOBILE Cloud Computing (MCC), with its abundant, configurable, and shareable resources, was once perceived as an efficient solution to the challenges posed by the computation-intensive applications from mobile user devices (UDs). UDs can offload the computation tasks to the cloud center, enabling efficient data processing in MCC. However, the primary limitation of MCC arises from the protracted latency in data transmission across backbone networks, posing a significant impediment to time-critical applications such as healthcare monitoring, autonomous driving, and real-time gaming. Then, Mobile Edge Computing (MEC) is proposed to address this inherent limitation in MCC, by offloading the computation tasks from UDs to the edge server (ES) for execution. This computing paradigm not only brings resources closer to UDs but also substantially decreases the latency in data transmission [1], [2]. MEC optimizes the efficiency of task processing in terms of both time and energy consumption, by allowing for real-time interactions and minimizing network latency.

The majority of studies within the MEC framework concentrate on enhancing computation offloading performance through the optimization of task offloading and resource allocation strategies. Despite massive efforts to improve the performance of MEC systems, several challenges remain that require urgent attention. For example, tasks generated by UDs frequently demonstrate heterogeneity in terms of input data size, computing resource demands, and latency constraints. An optimal resource scheduling strategy at the network edge must address the fierce competition for bandwidth and computing resources among numerous tasks [3]. Furthermore, a computation task generated by a UD typically comprises two distinct components: user-specific data and associated databases/libraries. The latter component often significantly surpasses the former in size, particularly for some AI-based applications that process large datasets. Offloading such tasks usually results in substantial transmission delays over the fronthaul network, thereby degrading the MEC performance.

Last but not least, the issue of user fairness is often ignored during task offloading in MEC. As a pivotal performance metric, user fairness not only fosters the motivation for task offloading in MEC, but also ensures equitable access to computing and communication resources of the edge server among heterogeneous UDs. Unfair resource allocation could lead to serious consequences for disadvantaged UDs in time-critical scenarios. For example, in the realm of online multi-player games, substantial disparities in latency can undermine the equity of competitive gaming, resulting in an unjust advantage for players with lower latency connections. Current works [4], [5], [6] focus on user fairness in MEC environments from various perspectives, but often overlook the primary motivation for introducing the MEC paradigm, which is to reduce the latency of offloaded tasks.

Received 29 August 2024; revised 12 May 2025; accepted 12 June 2025. Date of publication 4 July 2025; date of current version 8 August 2025. *(Corresponding author: Shuo Xiao.)*

Chaogang Tang, Yao Ding, and Shuo Xiao are with the School of Computer Science and Technology, China University of Mining and Technology, Xuzhou 221116, China (e-mail: cgtang@cumt.edu.cn; yding32@cumt.edu.cn; sxiao@cumt.edu.cn).

Zhenzhen Huang is with the School of Computer Science and Technology, China University of Mining and Technology, Xuzhou 221116, China, and also with the Library, China University of Mining and Technology, Xuzhou 221116, China (e-mail: huangzhenzhen@cumt.edu.cn).

Huaming Wu is with the Center for Applied Mathematics, Tianjin University, Tianjin 300072, China (e-mail: whming@tju.edu.cn).

Digital Object Identifier 10.1109/TSC.2025.3586093

Therefore, MEC systems should strive to balance the processing latency of latency-sensitive tasks with equitable resource allocation, thereby preventing service monopolization by specific users or applications.

To address the aforementioned issues, we introduce a fairness-aware caching-assisted edge computing framework in this paper. On the one hand, we cache the second component of computation task, i.e., task-associated databases/libraries, also referred to as services, at the ES/edge node (EN). Thus, only the user-specific data in computation tasks need to be offloaded in MEC, thereby substantially reducing the volume of transmitted data and effectively minimizing transmission latency [7]. On the other hand, we not only prioritize the minimization of task completion latency, but also place significant emphasis on ensuring fairness among users. Specifically, we employ a logarithmic function that uses the the difference between the maximal tolerable latency and the actual task completion time as its parameter to achieve proportional fairness. This logarithmic function exhibits desirable mathematical properties while aligning with the original intent of the MEC paradigm.

The goal of this paper is to jointly optimize service caching, task scheduling, and resource allocation strategies in MEC, and we formulate the optimization problem as a mixed-integer nonlinear programming (MINLP) problem. To solve this NP-hard optimization problem, we transform the origin problem into two distinct subproblems – the service caching and task offloading problem, and the multi-resource allocation problem, respectively. Then, the Advantage Actor-Critic (A2C) based approach is proposed to address the former problem, while a Lagrangian duality-based approach is adopted to tackle the latter problem. Furthermore, we integrate the proportional fairness criteria [8] by the logarithmic function to attain fairness goals.

The main contributions of this paper are listed as follows:

- We integrate the fairness issue into the MEC system, given the critical role of fairness in ensuring equitable access to computing and communication resources among heterogeneous UDs. Specifically, we propose using the difference between the maximal tolerable latency and the actual task completion time as a fairness metric to achieve proportional fairness.
- For the multi-service and multi-user edge computing scenario with cache assistance, we propose a collaborative service caching, task scheduling, and multi-resource allocation problem in MEC. This problem takes into consideration task deadlines, energy constraints, and various resource limitations, aiming to maximize the overall user satisfaction.
- The optimization problem is first decoupled. Then we introduced an A2C-based algorithm with the aim to learn and approximate near-optimal solutions for stochastic scheduling problems, particularly in caching and offloading decisions. In the following steps, we adopt Lagrangian duality-based methods to determine optimal resource allocation policy for offloaded tasks in ES.
- Through extensive simulation experiments, we compared the performance of several baseline strategies with our proposed solution. The results demonstrated the significant superiority of our approach.

The remainder of this paper is organized as follows: Section II reviews and discusses related research in the caching domain. Section III describes the system model, while Section IV defines the problem formulation. Section V introduces a multi-resource allocation algorithm based on Lagrangian dual theory. Section VI presents and demonstrates the A2C-based caching and task offloading model. Section VII provides the performance evaluation results of the proposed methods, and Section VIII summarizes the main conclusions and discusses future research directions.

II. Related Works

Compared to conventional MEC, cache-assisted task offloading offers significant potential for optimizing energy consumption, response time, and other performance metrics, making it a key research area in mobile edge computing [24], [25]. Additionally, user fairness, as a key performance metric for assessing the satisfaction level of users, should not be neglected during task offloading in MEC systems. In this section, we review the most relevant studies, analyze their contributions, and compare them with our work presented in this paper.

A. Fairness-Aware Optimization for MEC

Considering the complex computing demands of power Internet of Things devices, Li et al. [5] proposed a two-layer edge computing framework consisting of an edge server layer and an access point layer. Specifically, a fairness mechanism based on the Theil index was introduced to achieve balanced allocation within the system. The Industrial Internet of Things (IIoT) devices rely on computing resources to facilitate efficient data exchange and task offloading. In [10], the authors proposed a fairness-aware task offloading approach by formulating a multi-constraint optimization problem and solving it using a weighted max-min fairness algorithm.

Scheduling fairness can significantly impact the average QoS of vehicles in the vehicular edge computing paradigm. To address this, a dependent task offloading scheduling algorithm was proposed in [11], where scheduling fairness was integrated into the constraints. Specifically, a task sorting algorithm was utilized to determine the task scheduling sequence. Subsequently, a Q-learning-based approach was employed to solve the task offloading problem. Finally, scheduling fairness was ensured through a back-adjustment mechanism.

In an uncrewed aerial vehicles (UAVs)-assisted MEC system, UAVs can serve as computing nodes for offloading tasks from ground users. To ensure fairness, where each ground user has an equal opportunity to offload tasks, Karmakar et al. [12] proposed a fairness-aware secure task offloading approach. They utilized proportional fairness as the fairness metric and aimed to minimize the secrecy rate using SARSA-based deep reinforcement learning (DRL) technology. Extensive simulations demonstrated the performance of their approach. UAVs can also be used to shorten the latency in computation tasks in disaster scenarios. For instance, Wang et al. [13] explored task offloading in the given scenario and proposed a practical RIS phase shift mechanism to ensure fairness among ground terminals. Specifically, they employed a deep deterministic policy gradient

(DDPG) algorithm to address the formulated optimization problem. In another study, Fu et al. [14] emphasized energy efficiency (EE) fairness among IoT nodes within a UAV-assisted MEC network. To achieve this, they formulated a joint optimization problem involving multiple decision variables, aiming to enhance the computation EE of the least efficient IoT node.

Authors in [15] formulated a MINLP problem to minimize the maximum weighted energy while ensuring the fairness in IRS-assisted MEC system. A bisection search based approach was adopted to solve the subproblems. Extensive simulations were conducted to evaluate the performance of the MEC system, and the simulation results proved their advantages compared to several benchmark schemes.

B. Service Caching Assisted Task Offloading Optimization

In [16], Dai et al. combined task offloading with service caching in a cloud-fog scenario. Their goal was to make offloading decisions through non-cooperative games and dynamically cache services and adjust offloading decisions based on service popularity, thereby minimizing task delay and local energy consumption. In a multi-cell MEC system, Xu et al. [17] formulated the joint caching and offloading problem with the aim of minimizing computation latency. They devised an online algorithm based on Lyapunov optimization, which incorporates a decentralized approach using a variant of Gibbs sampling to facilitate distributed coordination among base stations.

Tang et al. [18] focused on application-oriented caching in vehicular edge computing, formulating an optimization problem to minimize the average response time. Due to the complexity of addressing this optimization challenge, they propose a Lyapunov-based greedy heuristic to derive the caching scheme across a long time horizon. The work in [19] focuses on service placement, task scheduling, and resource allocation in cloud-edge collaborative systems, aiming to minimize the overall task delay. The study simplifies the problem using Lyapunov optimization principles, transforming it into a deterministic problem for each time slot, and develops an iterative algorithm to comprehensively address the problem.

Moreover, researchers continue to explore the utilization of reinforcement learning techniques to achieve more efficient resource utilization and task processing. Wang et al. [20] detailed a methodology that employs asynchronous Advantage Actor-Critic design to simultaneously optimize service caching, computation offloading, and resource allocation within a three-tier edge-cloud framework. The aim is to cater to the latency constraints of mobile users while minimizing the costs associated with cloud service centers. In [21], Yan et al. introduced a cloud-edge computing framework for the Intelligent Meteorological System, focusing on service caching. They derived the service caching strategy using deep deterministic policy gradient (DDPG) to achieve optimal service coverage rates and minimal processing latency.

Tang et al. in [22] proposed a fairness-aware long-term cache sharing mechanism to promote the sharing of caching services in cloud computing environments. Specifically, their approach aimed to optimize cache usage efficiency while ensuring system fairness. This policy was implemented in Alluxio, and simulation results demonstrated that the strategy effectively maximized cache utilization while maintaining fair resource allocation across users. Chen et al. [23] highlighted the demands of end users and investigated a digital twin service caching and request routing problem. An optimization problem was formulated to maximize the minimum task completion ratio while ensuring fairness among MEC servers. An online algorithm was proposed to address this problem, and extensive simulations were conducted to demonstrate its effectiveness and efficiency.

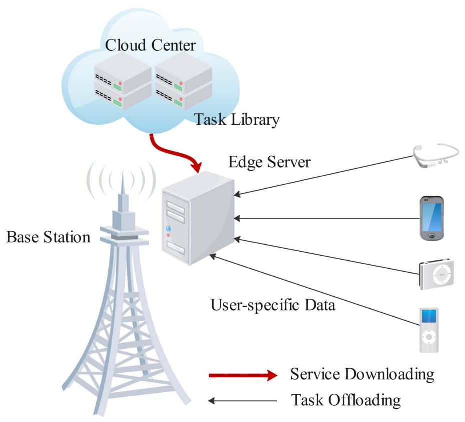


Fig. 1. System model.

C. Summary and Comparison

The aforementioned studies on fairness-aware optimization and service caching in MEC systems have made significant progress; however, they are limited by inherent constraints that hinder their effectiveness in dynamic, multi-service MEC environments. On the one hand, fairness metrics are introduced to balance resource allocation among users, but these approaches often neglect the interplay between task deadlines and energy consumption constraints. On the other hand, many works assume static service caching, which not only fails to adapt to the high dynamics of task arrivals but also underestimates the long-term costs associated with service replacement. Specifically, the comparison between related works and ours is summaried in Table I.

Overall, these studies either oversimplify system dynamics or fail to integrate fairness considerations with caching-offloading interactions. Our contribution lies in addressing these gaps through a unified, fairness-aware optimization framework that demonstrates proven scalability and efficiency. Specifically, we aim to integrate service caching, task scheduling, bandwidth resource allocation, and computing resource allocation into a cohesive optimization framework while ensuring fairness among users. Compared to existing studies that focus on only one or two dimensions of the aforementioned optimization aspects, our approach offers a more comprehensive optimization across all these dimensions.

III. System Model

In the depicted scenario shown in Fig. 1, the system model comprises a remote cloud, an ES, and multiple user devices.

TABLE I
EXISTING LITERATURE COMPARISON

Works	Caching	Fairness metric	Optimization objective	Algorithm
Xiao et al. [9]	×	QoE-fairness	QoE maximization	Soft AC algorithm
Li et al. [5]	×	Time average difference model	Allocation difference minimization	Lyapunov technology
Peng et al. [10]	×	Resource allocation fairness	Time and energy consumption	Weighted max-min fairness
Bi et al. [11]	×	Scheduling fairness	Application completion time	Q-learning algorithm
Karmakar et al. [12]	×	The proportional fairness	Secrecy rate minimization	SARSA-based DRL
Wang et al. [13]	×	RIS phase shift fairness	The system delay	DDPO-based algorithm
Fu et al. [14]	×	Energy efficiency (EE) fairness	The computation EE	BCD method
Wang et al. [15]	×	IRS fairness	Weighted consumed energy	Bisection search method
Dai et al. [16]	✓	×	Task latency and energy consumption	Non-cooperative gaming
Xu et al. [17]	✓	×	Computation delay	Lyapunov optimization
Tang et al. [18]	✓	×	Delay and energy consumption	Greedy approach
Tang et al. [7]	✓	×	Processing latency	Lyapunov optimization
Fan et al. [19]	✓	×	Overall task delay	Lyapunov optimization
Wang et al. [20]	✓	×	Overall task delay	A2C algorithm
Yan et al. [21]	✓	×	Total system costs	DDPG algorithm
Tang et al. [22]	✓	Cache sharing fairness	Cache usage efficiency	Alluxio based algorithm design
Chen et al. [23]	✓	Collaboration fairness	Minimum task completion ratio	Online greedy algorithm
Ours	✓	The proportional fairness	Task completion time variance	A2C algorithm

The ES, equipped with caching and computation capabilities, is positioned alongside the base station (BS), enabling potential task offloading from UDs to the ES through wireless networks. The optimization period $\mathcal{T}$ is segmented into distinct intervals referred to as time slots, i.e., $\mathcal{T} = \{1, 2, \ldots, T\}$. The set of UDs, is indexed by $\mathcal{N} = \{1, 2, \ldots, N\}$, capable of generating computation-intensive tasks. Assume that each UD n generates at most one computation task in each time slot. The task is assumed to be atomic and indivisible, and can be described by the input data size, computing resource demands, and delay constraints. Particularly, a task t_n^t generated by UD n in time slot t can be expressed as $t_n^t = (d_n^t, s_n^t, l_n^{max,t})$, where d_n^t denotes the input data size of the task, s_n^t represents the required computing resources (CPU cycles) for the task, and $l_n^{max,t}$ indicates the maximum tolerable delay for the task.

As previously mentioned, each computation task is associated with a corresponding service (i.e., the second component). The set of services is indexed by $\mathcal{A} = \{a_1, a_2, \ldots, a_K\}$ and K is the number of services that can be cached at the edge. Each service k has different storage requirement h_k. It should be noted that a single service can be associated with multiple computation tasks. Define $\alpha_{n,k}^t \in \{0, 1\}$ to indicate the correspondence between task n and service k in time slot t. $\alpha_{n,k}^t = 1$ indicates that the task t_n^t requests service k in slot t, and $\alpha_{n,k}^t = 0$, otherwise. $\delta(t) = \{\delta_1^t, \delta_2^t, \ldots, \delta_K^t\}$ denotes the caching profile of K services in time slot t, where $\delta_k^t \in \{0, 1\}$ is a binary variable. $\delta_k^t = 1$ denotes that service k is cached at the edge and $\delta_k^t = 0$ denotes that service k is not cached in time slot t. Thus, we have $\sum_{k\in\mathcal{K}} \delta_k^t h_k \leq C$, wherein C denotes the maximum storage capacity achievable at the ES.

Let $\beta_n^t = \sum_{k\in\mathcal{K}} \alpha_{n,k}^t \delta_k^t$ denote whether the service needed by task t_n^t is cached in time slot t. For instance, $\beta_n^t = 1$ denotes the service associated with t_n^t is cached at the ES, and $\beta_n^t = 0$ denotes the service associated with t_n^t is not cached. Each task can be processed locally, and only those tasks with their associated services cached at the edge can be offloaded for execution. Thus, define a binary offloading decision variable $\lambda_n^t \in \{0, 1\}$, where $\lambda_n^t = 0$ indicates that the task is executed locally, and $\lambda_n^t = 1$ indicates that the task is offloaded to the edge for execution.

At the beginning of each time slot, UDs which generate tasks will disseminate the requests, along with beacon information, to the ES. Given the caching profile in the previous time slot and designed evaluation metric, ES determines which services should be cached and which tasks should be offloaded. It should be noted that the ES can download and cache services from a remote cloud, in accordance with an established cache update policy. ES starts to execute a task only when both its required service and input data are available.

A. Task Processed Locally

Let f_n^l denote the average processing frequency (in CPU cycles per second) of UD n for local computation. Thus, the total processing time of task n for local execution is expressed as:

$$L_l^t = s_n^t / f_n^l. \tag{1}$$

B. Task Offloading to the Edge

The Dynamic Spectrum Access (DSA) technology can efficiently allocate spectrum resources among UDs, while satisfying the diverse communication requirements of different UDs in realtime scenarios [26]. Define $B(t) = \{b_1^t, b_2^t, \ldots, b_N^t\}$ as a bandwidth distribution vector, indicating the spectrum resources assigned to each task in a single time slot. Hence, the inequation $\sum_{n\in\mathcal{N}} \lambda_n^t b_n^t \leq W$ holds, since the communication resources allocation across all offloaded tasks should not surpass the total uplink bandwidth capacity W.

The latency associated with transmitting input data from UD n to the ES via the wireless communication channel can be mathematically expressed as:

$$l_{n,trs}^{e,t} = d_n^t / r_n^{e,t}, \tag{2}$$

where $r_n^{e,t}$ represents the achievable uplink data rate, which is expressed as:

$$r_n^{e,t} = b_n^t \log\left(1 + \frac{g_n^t p_n}{N_0 b_n^t}\right), \tag{3}$$

where p_n signifies the transmission power of UD n, g_n^t represents the wireless channel gain between UD n and the ES during time slot t, and N_0 denotes the spectral density of noise power.

If a task can be offloaded to the ES for execution, the requested service k must be accessible at the ES. If this service was not stored in the ES during the previous time interval, it must be downloaded by the ES from the remote cloud. Denote by $r^{c,t}$ the transmission rate between the ES and the remote cloud. Then, the time taken for downloading service k from the cloud to the ES is

$$l_{n,ser}^{c,t} = \frac{\max(\beta_n^t - \beta_n^{t-1}, 0)\sum_{k=1}^K \alpha_{n,k}^t h_k}{r^{c,t}}, \tag{4}$$

where $\beta_n^t - \beta_n^{t-1}$ can indicate the caching update decision for the required service in time slot t. When $\beta_n^t - \beta_n^{t-1} = 1$, it means that the required service was not cached in the prior time interval but will be included in current time slot t. $\beta_n^t - \beta_n^{t-1} = 0$ means that the caching status of the required service remains the same across time slot t and $t-1$. When $\beta_n^t - \beta_n^{t-1} = -1$, it denotes that the necessary service was cached during previous time interval but will be removed in time slot t.

Define the vector $f^e(t) = \{f_1^{e,t}, f_2^{e,t}, \ldots, f_N^{e,t}\}$ to denote the allocation of computing resources among the tasks offloaded to the ES during time slot t, where $f_n^{e,t}$ represents the amount of CPU-cycle frequency assigned to accomplish a particular computation task. Denote by F_e the total computing resources available at the ES. Thus, the computing resources $f_n^{e,t}$ ($\forall n \in \mathcal{N}$) should satisfy the constraint: $\sum_{n\in\mathcal{N}} \lambda_n^t f_n^{e,t} \leq F_e$. Then, the execution latency for task t_n^t can be described as:

$$l_{n,exe}^{e,t} = s_n^t / f_n^{e,t}. \tag{5}$$

The overall completion time of offloading task can be summarized as:

$$L_e^t = \max(l_{n,trs}^{e,t}, l_{n,ser}^{c,t}) + l_{n,exe}^{e,t}. \tag{6}$$

The energy consumption for executing the task at ES can be formulated as:

$$e_{n,exe}^{e,t} = k^e s_n^t f_n^{e,t2}, \tag{7}$$

where k^e signifies the effective switched capacitance coefficient associated with the ES. Additionally, the total storage consumption required for caching selected services within the ES is represented by $e_{str}^{e,t} = \sum_{k\in\mathcal{K}} \delta_k^t \gamma_k$, where γ_k denotes the average static power consumption for caching service k. The total energy consumption for accomplish offloading tasks within time slot t is calculated as:

$$E_e^t = e_{str}^{e,t} + \sum_{n=1}^N \lambda_n^t e_{n,exe}^{e,t}. \tag{8}$$

The symbols employed within this article are concisely compiled and summarized in Table II for reference and clarity.

TABLE II
NOTATIONS AND DESCRIPTIONS

N	The number of user devices
K	The number of services
d_n^t	The input data size of task n
s_n^t	The computing resource required for task n
$l_n^{max,t}$	The maximal tolerable latency of task n
b_n^t	The allocated communication resource of task n
W	The total uplink bandwidth of the ES
$f_n^{e,t}$	The allocated computing resource of task n
F_e	The total processing capacity of the ES
h_k	The storage requirement of service k
C	The maximum storage capacity of the ES
δ_k^t	The caching decision of service k
λ_n^t	The offloading decision of task n
β_n^t	The caching state of the required service for task n

IV. Problem Formulation

At the beginning, the satisfaction function is defined, encompassing both system response latency and fairness among users, thereby offering a comprehensive evaluation of the overall user experience in the task offloading scenario. Following other works [27], [28], a logarithmic function is also employed to achieve proportional fairness. The satisfaction function for completing task t_n^t is defined as:

$$S_n^t = \log(1 + \phi + l_n^{max,t} - (\lambda_n^t L_e^t + (1-\lambda_n^t)L_l^t)), \tag{9}$$

where ϕ calibrates the satisfaction metric to ensure the non-negativity.

Our objective is to maximize the average satisfaction of all tasks while adhering to the constraints on task completion delay and energy consumption. The optimization problem is formulated as:

$$P1: \max_{\delta(t),\lambda(t),f^e(t),B(t)} \lim_{T\to\infty} \frac{1}{T} \sum\nolimits_{t=1}^T \sum\nolimits_{n\in\mathcal{N}} S_n^t \tag{10}$$

$$\text{s.t.} \quad \lambda_n^t L_e^t + (1-\lambda_n^t)L_l^t \leq l_n^{max,t}, \quad \forall n \in \mathcal{N}, \forall t \in \mathcal{T}, \tag{10a}$$

$$E_e^t \leq Q, \quad \forall t \in \mathcal{T}, \tag{10b}$$

$$\sum_{k\in\mathcal{K}} \delta_k^t h_k^t \leq C, \quad \forall t \in \mathcal{T}, \tag{10c}$$

$$\sum_{n\in\mathcal{N}} \lambda_n^t b_n^t \leq W, \quad \forall t \in \mathcal{T}, \tag{10d}$$

$$\sum_{n\in\mathcal{N}} \lambda_n^t f_n^{e,t} \leq F_e, \quad \forall t \in \mathcal{T}, \tag{10e}$$

$$b_n^t \geq 0, \quad \forall n \in \mathcal{N}, \ \forall t \in \mathcal{T}, \tag{10f}$$

$$f_n^{e,t} \geq 0, \quad \forall n \in \mathcal{N}, \ \forall t \in \mathcal{T}, \tag{10g}$$

$$\delta_k^t \in \{0,1\}, \quad \forall n \in \mathcal{N}, \ \forall t \in \mathcal{T}, \tag{10h}$$

$$\lambda_n^t \in \{0,1\}, \quad \forall n \in \mathcal{N}, \ \forall t \in \mathcal{T}, \tag{10i}$$

where condition (10b) specifies that the completion delay for each task should not surpass its permissible latency limit, while (10c) imposes a limit on the energy consumption of tasks on the ES, ensuring it does not exceed the predefined threshold. Constraint (10d) ensures that the storage requirement of cached services during one time slot remain within the maximal storage capacity of the ES. Conditions (10e) and (10f) implies that the total computation and communication capacity allocated across all tasks during the time slot t should not exceed the total resources available within the ES.

Remark: The optimization problem $P1$ is a prototypical mixed-integer nonlinear programming (MINLP) problem, and the computational cost of obtaining an optimal solution in large-scale MEC environment is excessively high. The interdependencies between tasks and resources further complicate the solution space, resulting in an exponential increase in computational requirements as the problem size expands. To address these challenges, we partition the original problem into two subproblems: the service caching and task offloading (SC&TO) problem, and the multi-resource allocation (MRA) problem, which are addressed separately in Sections Sections VI and V, respectively.

V. Multi-Resource Allocation Algorithm

Examining the formulation of problem $P1$, it is evident that the resource allocation decision at the ES in time slot t can only impact task completion delay and energy consumption at the ES. Thus, given a SC&TO strategy in time slot t, we only need to focus on MRA problem in the slot t. To streamline the MRA problem, we decompose it into two sub-problems: a bandwidth resource allocation (BRA) problem and a CPU frequency allocation (CRA) problem. The two subproblems can be addressed in an alternative and iterative manner until convergence is achieved.

A. Communication Resource Allocation Problem

Based on the preceding analysis, given caching, offloading, and CPU frequency allocation decisions in a single time slot, the BRA problem assumes the following structure,

$$P1.1: \quad \max_{B(t)} \sum\nolimits_{n\in\mathcal{N}} S_n^t$$
$$\text{(10a)}, \text{(10d)}, \text{(10f)}. \tag{11}$$

Lemma 1: Within the optimal solution set of problem $P1.1$, there always exists a solution such that $l_{n,trs}^{e,t} \geq l_{n,ser}^{c,t}$ holds for all the offloaded tasks.

Proof: First, it is obvious that when $b_n^t > 0, l_{n,trs}^{e,t} = d_n^t / r_n^{e,t}$ decreases monotonically with respect to b_n^t. Due to space limitations, the proof is not included here.

Let $\boldsymbol{b}^{t\star}$ be an element in the set of optimal solutions to problem $P1.1$.

Case 1: The requested services for the offloaded tasks were all cached in the previous time slot, then the inequality $l_{n,trs}^{e,t} \geq l_{n,ser}^{c,t}$ holds for all the offloaded tasks.

Case 2: There exists a task t_n^t $(n \in \mathcal{N})$ requesting a service that is not cached in the ES, and the inequality $l_{n,trs}^{e,t} < l_{n,ser}^{c,t}$ holds. In the meanwhile, there exists an offloading task $t_m^t \in \mathcal{N}$ whose requested service was cached in the ES in the previous time slot $t-1$. $l_{n,trs}^{e,t}$ is monotonically decreasing with respect to b_n^t, and we can slightly decrease $b_n^{t\,\star}$ to $b_n^{t\,\star\prime}$, i.e., $b_n^{t\,\star\prime} = b_n^{t\,\star} - \epsilon$, where ϵ is a small positive number to guarantee that $b_n^{t\,\star\prime}$ satisfies the inequality $l_{n,trs}^{e,t} < l_{n,ser}^{c,t}$. Then, this communication resource surplus ϵ can be allocated to task t_m^t. By doing so, it is obvious that the value for the optimization objective of $P1.1$ increases, so we can find a better optimal solution $b_n^{t\,\star\prime}$ in contrast to $b_n^{t\,\star}$. It contradicts the assumption that $b_m^{t\,\star}$ is one of optimal solutions to problem $P1.1$. Therefore, the inequality $l_{n,trs}^{e,t} \geq l_{n,ser}^{c,t}$ holds true for all offloading tasks in this case.

Case 3: None of the requested services for offloaded tasks are cached in the ES, and there exists a task t_n^t $(\forall n \in \mathcal{N})$ that satisfies the inequality $l_{n,trs}^{e,t} < l_{n,ser}^{c,t}$. Similarly, we can slightly decrease $b_n^{t\,\star}$ to $b_n^{t\,\star\prime}$, i.e., $b_n^{t\,\star\prime} = b_n^{t\,\star} - \epsilon$, where ϵ is a small positive number such that $b_n^{t\,\star\prime}$ satisfies the equality $l_{n,trs}^{e,t} = l_{n,ser}^{c,t}$. In this case, the optimization objective of $P1.1$ remains unchanged. Thus, there exists an optimal solution where the inequality $l_{n,trs}^{e,t} \geq l_{n,ser}^{c,t}$ holds true for all the offloaded tasks.

Based on Lemma 1, we can reframe problem $P1.1$ as $P1.1.1$, with the optimal solution set of $P1.1.1$ being a subset of the optimal solution set of $P1.1$,

$$P1.1.1: \quad \max_{B(t)} \sum\nolimits_{n\in\mathcal{N}} S_n^t \tag{12}$$
$$\text{s.t.} \quad l_{n,trs}^{e,t} \geq l_{n,ser}^{c,t},$$
$$\text{(10a)}, \text{(10d)}, \text{(10f)}. \tag{12a}$$

Lemma 2: The following equivalent problem can be expressed by reformulating problem $P1.1.1$ as:

$$P1.1.1: \quad \max_{B(t)} \sum\nolimits_{n\in\mathcal{N}} S_n^{t\,\prime} \tag{13}$$
$$\text{s.t.} \quad b_n^t \geq b_n^{min,t}, \quad \forall n \in \mathcal{N}, \forall t \in \mathcal{T}, \tag{13a}$$
$$b_n^t \leq b_n^{max,t}, \quad \forall n \in \mathcal{N}, \forall t \in \mathcal{T},$$
$$\text{(10d)}. \tag{13b}$$

Here,

$$S_n^{t\,\prime} = 1 + \phi + l_n^{max,t} - \left(\lambda_n^t \left(\frac{d_n^t}{r_n^{e,t}} + l_{exe}^{e,t}\right) + (1-\lambda_n^t) L_l^t\right), \tag{14}$$

$$b_n^{min,t} = \begin{cases} H\left(l_n^{max,t} - \frac{s_n^t}{f_n^{e,t}}\right), & \lambda_n^t = 1; \\ 0, & \lambda_n^t = 0, \end{cases} \tag{15}$$

where

$$H\left(l_n^{max,t} - \frac{s_n^t}{f_n^{e,t}}\right) = -1 \Bigg/ \left(\left(W_{-1}\left(\frac{-d_n^t \ln 2 N_0}{(l_n^{max,t} - \frac{s_n^t}{f_n^{e,t}}) p_n^e g_n^t} e^{\frac{-d_n^t \ln 2 N_0}{(l_n^{max,t} - \frac{s_n^t}{f_n^{e,t}}) p_n^e g_n^t}}\right) + \frac{d_n^t \ln 2 N_0}{(l_n^{max,t} - \frac{s_n^t}{f_n^{e,t}}) p_n^e g_n^t}\right) \frac{l_n^{max,t} - \frac{s_n^t}{f_n^{e,t}}}{d_n^t \ln 2}\right), \tag{16}$$

and

$$b_n^{max,t} = \begin{cases} H(l_{n,ser}^{c,t}), & \beta_n^t - \beta_n^{t-1} = 1, \lambda_n^t = 1; \\ W, & \beta_n^t - \beta_n^{t-1} = 0, \lambda_n^t = 1; \\ 0, & \text{others}, \end{cases} \tag{17}$$

where

$$H(l_{n,ser}^{c,t}) = -1 \Bigg/ \left(\left(W_{-1} \left(\frac{-d_n^t \ln 2N_0}{l_{n,ser}^{c,t} p_n^e g_n^t} e^{\frac{-d_n^t \ln 2N_0}{l_{n,ser}^{c,t} p_n^e g_n^t}} \right. \right. \right. \\ \left. \left. + \frac{d_n^t \ln 2N_0}{l_{n,ser}^{c,t} p_n^e g_n^t} \right) \frac{l_{n,ser}^{c,t}}{d_n^t \ln 2} \right), \tag{18}$$

and $W_{-1}(\cdot)$ is Lambert W function.

Proof: Due to $l_{n,trs}^{e,t} \geq l_{n,ser}^{c,t}$, S_n^t can be simplified to $S_n^{t\prime}$. To facilitate calculations, constraints (10b), (10g) have been simplified to (13a), where $b_n^{min,t}$ is the minimal computing resource needed to satisfy constraint (10b) for those offloaded tasks, and $b_n^{max,t}$ is the maximum bandwidth allocation satisfying $l_{n,trs}^{e,t} = l_{n,ser}^{c,t}$ for these offloaded tasks whose required service has not been cached in the ES.

Consider the following equation,

$$l_{n,trs}^{e,t} = \frac{d_n^t}{r_n^{e,t}} = \frac{d_n^t}{b_n^t \log_2 (1 + \frac{g_n^t p_n}{N_0 b_n^t})} \triangleq C_0. \tag{19}$$

That is,

$$\frac{d_n^t ln2}{C_0 b_n^t} = \ln \left(1 + \frac{g_n^t p_n}{N_0 b_n^t} \right). \tag{20}$$

By applying the natural exponential operation and certain transformations to both sides, we have

$$\left(-\frac{d_n^t ln2N_0}{C_0 p_n^{t,e} g_n^t} - \frac{d_n^t ln2}{C_0 b_n^t} \right) e^{-\frac{d_n^t ln2N_0}{C_0 p_n^{t,e} g_n^t} - \frac{d_n^t ln2}{C_0 b_n^t}} \\ = -\frac{d_n^t ln2N_0}{C_0 p_n^{t,e} g_n^t} e^{-\frac{d_n^t ln2N_0}{C_0 p_n^{t,e} g_n^t}}. \tag{21}$$

Let $x = -\frac{d_n^t ln2N_0}{C_0 p_n^{t,e} g_n^t} - \frac{d_n^t ln2}{C_0 b_n^t}$, the above expression becomes

$$x e^x = -\frac{d_n^t ln2N_0}{C_0 p_n^{t,e} g_n^t} e^{-\frac{d_n^t ln2N_0}{C_0 p_n^{t,e} g_n^t}}. \tag{22}$$

Consequently, we can solve the equation for x by

$$x = W_{-1} \left(-\frac{d_n^t ln2N_0}{C_0 p_n^{t,e} g_n^t} e^{-\frac{d_n^t ln2N_0}{C_0 p_n^{t,e} g_n^t}} \right), \tag{23}$$

and b_n^t can be expressed as

$$b_n^t = H(C_0) = -1 \Bigg/ \left(\left(W_{-1} \left(-\frac{d_n^t ln2N_0}{C_0 p_n^{t,e} g_n^t} e^{-\frac{d_n^t ln2N_0}{C_0 p_n^{t,e} g_n^t}} \right) \right. \right. \\ \left. \left. + \frac{d_n^t ln2N_0}{C_0 p_n^{t,e} g_n^t} \right) \frac{C_0}{d_n^t ln2} \right). \tag{24}$$

For those offloaded tasks, $\frac{d_n^t}{r_n^{e,t}} \leq l_n^{max,t} - \frac{s_n^t}{f_n^{e,t}}$ holds, therefore, $b_n^{min,t} = H(l_n^{max,t} - \frac{s_n^t}{f_n^{e,t}})$. For these offloading tasks whose required service has not been cached at ES, $\frac{d_n^t}{r_n^{e,t}} \geq l_{n,ser}^{c,t}$, thus, $b_n^{max,t} = H(l_{n,ser}^{c,t})$.

Lemma 3: Given the caching decision $\delta(t)$, offloading scheme $\lambda(t)$, and computing resource allocation $f^e(t)$, respectively, $P1.1.1$ exhibits convexity w.r.t. $B(t)$.

Proof: First, we need to prove the concavity of the objective function with respect to b_n^t. Define $S_{n,in}^{t\prime}$ as the inner part of $S_n^{t\prime}$, i.e., ${S_{n,in}^t}' = 1 + \phi + l_n^{max,t} - (\lambda_n^t(\frac{d_n^t}{r_n^{e,t}} + l_{exe}^{e,t}) + (1 - \lambda_n^t) l_{tol}^{l,t})$. It is obvious that only the term $\frac{d_n^t}{r_n^{e,t}}$ is dependent on b_n^t. Thus, proving the concavity of $S_{n,in}^{t\prime}$ is equivalent to prove the convexity of $\frac{d_n^t}{r_n^{e,t}}$ w.r.t. b_n^t. Since

$$\frac{\partial^2 \frac{d_n^t}{r_n^{e,t}}}{\partial {b_n^t}^2} = d_n^t \ln 2 \left(\frac{2\left(\left(1 + \frac{g_n^t p_n}{N_0 b_n^t}\right) \ln \left(1 + \frac{g_n^t p_n}{N_0 b_n^t}\right) - \frac{g_n^t p_n}{N_0 b_n^t}\right)^2}{(b_n^t)^3 \left(1 + \frac{g_n^t p_n}{N_0 b_n^t}\right)^2 \ln \left(1 + \frac{g_n^t p_n}{N_0 b_n^t}\right)^3} \right. \\ \left. + \frac{\frac{g_n^t p_n}{N_0 b_n^t}^2 \ln \left(1 + \frac{g_n^t p_n}{N_0 b_n^t}\right)}{(b_n^t)^3 \left(1 + \frac{g_n^t p_n}{N_0 b_n^t}\right)^2 \ln \left(1 + \frac{g_n^t p_n}{N_0 b_n^t}\right)^3} \right) \geq 0, \tag{25}$$

$\frac{d_n^t}{r_n^{e,t}}$ is convex with respect to b_n^t, rendering the concavity of $S_{n,in}^{t\prime}$ with respect to b_n^t. Given the concave nature and non-decreasing properties of the logarithmic function, $\sum_{n \in \mathcal{N}} S_{n,in}^{t\prime}$ remains concave with respect to b_n^t . Moreover, considering that (13a), (13b) and (10e) are linear constraints, problem $P1.1.1$ is inherently convex.

Thus, the Lagrangian function takes the following form

$$L(B(t), \theta) = \sum_{n \in \mathcal{N}} {S_n^t}' - \theta \left(\sum_{n \in \mathcal{N}} \lambda_n^t b_n^t - W \right), \\ \text{(13a), (13b)}. \tag{26}$$

The optimal solution requirements for $P1.1.1$ are dictated by the Karush-Kuhn-Tucker conditions, as follows

$$\frac{\partial {S_n^t}'}{\partial b_n^t} - \lambda_n^t \theta = 0, \tag{27a}$$

$$\theta \left(\sum_{n \in \mathcal{N}} \lambda_n^t b_n^t - W \right) = 0, \\ \text{(10d), (13a), (13b)}. \tag{27b}$$

Lemma 4: From the equation (27a), it can be deduced that b_n^t monotonically decreases with respect to θ and θ monotonically decreases with respect to b_n^t.

Proof: Deriving both sides of the equation (27a) gives

$$\frac{\partial \frac{\partial {S_n^t}'}{\partial b_n^t}}{\partial \theta} - \frac{\partial \lambda_n^t \theta}{\partial \theta} = 0, \tag{28}$$

i.e.

$$\frac{\partial \frac{\partial {S_n^t}'}{\partial b_n^t}}{\partial b_n^t} \frac{\partial b_n^t}{\partial \theta} - \lambda_n^t = 0. \tag{29}$$

Then we have

$$\frac{\partial b_n^t}{\partial \theta} = \frac{\lambda_n^t}{\frac{\partial^2 S_n^{t\,\prime}}{\partial {b_n^t}^2}}. \tag{30}$$

Since only the term $\frac{-d_n^t}{r_n^{e,t}}$ in $S_n^{t\,\prime}$ is related to b_n^t, based on the chain rule of differentiation, the original expression can be rewritten as:

$$\frac{\partial b_n^t}{\partial \theta} = \frac{\lambda_n^t}{\frac{\partial^2 \frac{-d_n^t}{r_n^{e,t}}}{\partial {b_n^t}^2}}. \tag{31}$$

From Lemma 2, it follows that $\frac{\partial^2 \frac{d_n^t}{r_n^{e,t}}}{\partial {b_n^t}^2} \geq 0$. Additionally, since $\lambda_n^t \geq 0$, we conclude that $\frac{\partial b_n^t}{\partial \theta} \leq 0$. Consequently, b_n^t is monotonically decreasing with respect to θ. As the monotonicity of the inverse function is the same as that of the original function, θ is monotonically decreasing with respect to b_n^t.

With above analysis, we have proposed an efficient algorithm of problem $P1.1.1$. According to (27a), θ can be expressed as a function of b_n^t as $\theta(b_n^t)$. According to Lemma 3, the minimum and maximum values of θ for each task, denoted as $\theta_n^{\min,t} = \theta(b_n^{\max,t})$ and $\theta_n^{max,t} = \theta(b_n^{min,t})$. Meanwhile, the maximum and minimum values of θ are given by:

$$\theta^{min,t} = \max(\theta_n^{min,t}), \tag{32}$$

$$\theta^{max,t} = \min(\theta_n^{max,t}). \tag{33}$$

Step 1: When comparing θ with the respective minimum and maximum values $\theta_n^{min,t}$ and $\theta_n^{max,t}$ for each task, if θ falls within the range defined by $\theta_n^{min,t}$ and $\theta_n^{max,t}$, then the corresponding b_n^t will also lie between $b_n^{min,t}$ and $b_n^{max,t}$. Considering (27a), (13a), and (13b), the communication allocation variables b_n^t can be expressed as a function of θ

$$b_n^t = \begin{cases} b_n^{min,t}, & \theta < \theta_n^{min,t}; \\ b_n^{max,t}, & \theta > \theta_n^{max,t}; \\ b_n^t(\theta), & \text{otherwise}, \end{cases} \tag{34}$$

where $b_n^t(\theta)$ is the solution to (27a). Due to its monotonic nature, we can employ a binary search algorithm to determine b_n^t.

Step 2: Based on the $B(t)$ obtained from Step 1, assess whether it satisfies constraint (10e), and also employ the bisection method to update θ.

Step 3: Repeat Step 1 and 2 until θ converges. The detailed steps are outlined in Algorithm 1.

Remark: The algorithm optimizes the bandwidth b_n^t using a bisection search approach over the resource space bounded by the interval between θ_n^{min} and θ_n^{max}; Optimally determining θ_n^{min} and θ_n^{max} requires the time complexity of $O(N)$ where N is the number of total tasks in the worst case. For large N, the $O(N)$ per-iteration complexity of BRA is still acceptable. Furthermore, the adoption of parallelization techniques (e.g., GPU acceleration for gradient computations) and approximation methods can reduce the computational time.

Algorithm 1: Communication Resource Allocation Algorithm.

```
1:  Initialization:
2:  $\mathcal{O}$: the set of offloading tasks;
3:  $f^e(t)$: the computing resource allocation strategy for $\mathcal{O}$;
4:  $tol_b$, $tol_\theta$: the tolerance error for $B(t)$, $\theta$;
5:  Calculate $b_n^{min,t}$, $b_n^{max,t}$ based on (15), (17);
6:  repeat
7:    $\theta_n^{min} = \theta(b_n^{max,t})$;
8:    $\theta_n^{max} = \theta(b_n^{min,t})$;
9:    Calculate $\theta^{min}$, $\theta^{max}$ based on (32), (33);
10:   $\theta = (\theta^{max} + \theta^{min})/2$;
11:   for $n \in \mathcal{O}$ do
12:     if $\theta \leq \theta_n^{min}$ then
13:       $b_n^t = b_n^{max,t}$;
14:     else if $\theta \geq \theta_n^{max}$ then
15:       $b_n^t = b_n^{min,t}$;
16:     else
17:       repeat
18:         $b_n^t = (b_n^{min,t} + b_n^{max,t})/2$;
19:         if $f(b_n^t) = 0$ then
20:           break;
21:         else if $f(b_n^t) < 0$ then
22:         $b_n^{max,t} = b_n^t$;
23:       else
24:           $b_n^{min,t} = b_n^t$;
25:         end if
26:       until $|f(b_n^t)| \leq tol_b$
27:     end if
28:   end for
29:   if $\sum_{n\in\mathcal{O}} b_n^t > W$ then
30:     $\theta_{min} = \theta$;
31:     else
32:     $\theta_{max} = \theta$;
33:   end if
34: until $|\theta^{min} - \theta^{max}| < tol_\theta$
```

B. Computing Resource Allocation Algorithm

Given fixed caching, offloading, and bandwidth allocation decisions, the CRA problem is formulated as:

$$P1.2: \quad \max_{f^e(t)} \quad \sum\nolimits_{n\in\mathcal{N}} S_n^t \tag{35}$$

$$\text{s.t.} \quad f_n^{e,t} \geq f_n^{min,e,t}, \quad \forall n \in \mathcal{N},$$

$$(10b), (10e). \tag{35a}$$

Here, (10b), (10h) has been simplified to (35a), and $f_n^{min,e,t}$ is the minimum computing resource required to satisfy the latency constraint. Since the second-order derivative of $P1.2$ with respect to f_n^e, $\frac{\partial^2 S_n^t}{\partial {f_n^{e,t}}^2} \leq 0$, the optimization problem $P1.2$ with the presence of linear and convex constraints can be established as a convex problem. Thus $P1.2$ is addressed using the Lagrangian dual method, where primal variables are iteratively

Algorithm 2: Computing Resource Allocation Algorithm.

```
1:  Initialization:
2:  $\mathcal{O}$: the set of offloading tasks;
3:  $b(t)$: the communication resource allocation strategy
    for $\mathcal{O}$;
4:  $tol_{f^e}$: the tolerance error for $f_e(t)$;
5:  $i \longleftarrow 0$;
6:  repeat
7:     Update Lagrange multipliers $\nu_n$, $\mu$, and $\omega$ based on
       (37), respectively;
8:     Calculate $f^e(t)$ based on (38);
9:     $i \longleftarrow i+1$;
10: until $|f^e_n(i) - f^e_n(i-1)| < tol_{f^e}$
```

optimized with fixed Lagrange multipliers, and Lagrange multipliers are determined based on optimized primal variables. The Lagrangian function is expressed as:

$$L(f^e(t), \boldsymbol{\nu}, \mu, \omega) = \sum_{n\in\mathcal{N}} S^t_n - \sum_{n\in\mathcal{N}} \nu_n (f^{e,t}_{n,min} - f^{e,t}_n) - \mu \left(\sum_{n\in\mathcal{N}} \lambda^t_n k^e s^t_n f^{e,t2}_n + \sum_{k\in\mathcal{K}} \delta^t_k \gamma_k - Q \right) - \omega \left(\sum_{n\in\mathcal{N}} \lambda^t_n f^{e,t}_n - F_e \right), \tag{36}$$

where the coefficients $\boldsymbol{\nu}$, μ and ω represent Lagrange multipliers linked to minimum resource constraint (35a), energy threshold (10c), and upper resource constraint (10f), respectively. Specifically, the Lagrange multipliers are iteratively updated as follows:

$$\nu_n(i+1) = [\nu_n(i) + c_1(f^{e,t}_{n,min} - f^{e,t}_n)]^+,$$
$$\mu(i+1) = [\mu(i) + c_2(E^t_e - Q)]^+,$$
$$\omega(i+1) = \left[\omega(i) + c_3 \left(\sum_{n\in\mathcal{N}} \lambda^t_n f^{e,t}_n - F_e \right)\right]^+. \tag{37}$$

Here, $[\cdot]^+$ represents $\max(0,\cdot)$, while $c1, c2, c3$ stand for positive gradient step sizes, with i indicating the current iteration number in the optimization process. Following the determination of Lagrange multiplier values, $f^e(t)$ is computed by solving the following function,

$$\frac{\partial L}{\partial f^{e,t}_n} = \frac{1}{\ln 2 S^t_n} \frac{\lambda^t_n s^t_n}{f^{e2}_n} + \nu \frac{\lambda^t_n s^t_n}{f^{e2}_n} - 2\mu\lambda^t_n k^e s^t_n f^{e,t}_n - \omega\lambda^t_n = 0. \tag{38}$$

The details on solving the CRA problem is outlined in Algorithm 2. Note that Algorithms 1 and 2 are employed to manage communication and computing resource allocation separately. Through iterative resolution of these distinct challenges, we achieve optimal resource allocation based on certain service caching and task offloading state, as outlined in Algorithm 3.

Algorithm 3: Multi-Resource Allocation Algorithm.

```
1: Initialization:
2: $\mathcal{O}$: the set of offloading tasks;
3: Set feasible solutions for $b(t)$ and $f^e(t)$ randomly;
4: repeat
5:    Calculate $f^e(t)$ based on Algorithm 1;
6:    Calculate $B(t)$ based on Algorithm 2;
7: until Convergence condition satisfied
```

VI. Service Caching and Task Offloading Algorithm Based on RL Method

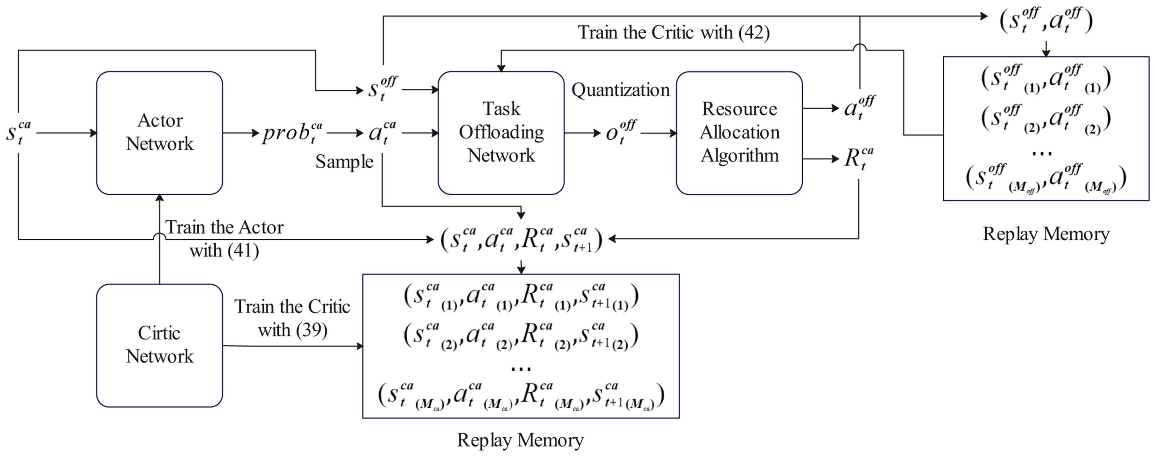
In this section, we adopt an A2C-based framework to make the service caching and task offloading decisions. The schematic depiction of the proposed ACSCTO algorithm is presented in Fig. 2, which comprises two main modules: the service caching module (SCM) and the task offloading module (TOM). The SCM is responsible for generating service caching decision, and then SCM relay the decision to TOM. The TOM makes task offloading decisions, based on received caching decision. Subsequently, using Algorithm 3, it computes the corresponding rewards and transfers them back to the SCM. The iterative process is repeated to further refine the strategies for service caching and task scheduling. The descriptions of these two modules will be provided in Section VI-A and VI-B.

A. Service Caching Module

1) Input Specification and Pre-Processing: Based on the A2C algorithm, the service caching module comprises two networks: the actor network and the critic network. Both networks share the same input, referred to as s^{ca}_t, as shown in Fig. 3, encompassing task information, service information, and the caching state from the previous time slot. Task information includes the required service number, task input data size, necessary CPU cycles, task deadline, and channel gain between each task and the ES. Service information encompasses service caching, energy consumption and storage requirements. Each feature of task t^t_n is transformed into an one-hot vector to highlight the correspondence between tasks and services, followed by the concatenation operation of all feature vectors.

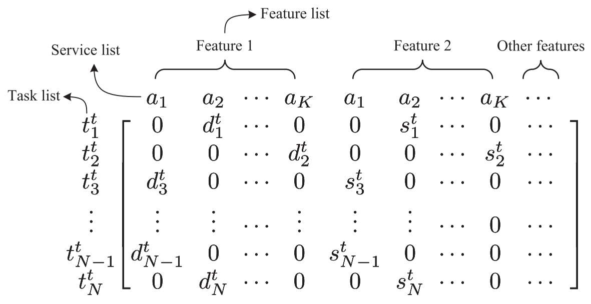
2) Network Architecture and Output Specification: Apart from the distinction in input and output outcomes, both the actor and critic networks demonstrate a similar network structure, with the depicted architecture of the critic network shown in Fig. 4.

The two-dimensional input s^{ca}_t in the critic network is initially flattened and then undergoes a group layer normalization process. This process can realize the normalization of heterogeneous features with the same mean and variance [29]. Then, we concatenate the output of the normalization layer with action a^{ca}_t, and the result can be obtained through dense layers. Given the limited storage capacity of the ES, the output of the actor network must satisfy the constraint $\sum_{k\in\mathcal{K}} \delta^t_n h_k \leq C$. Note that, creating a model with constrained output can bring about computational challenges [30], so we opt for an unconstrained definition of model action. The actor network's output undergoes



Train the Critic with (42)
(s_t^{off}, a_t^{off})
s_t^{off}
s_t^{ca}
Actor Network
$prob_t^{ca}$
a_t^{ca}
Sample
Task Offloading Network
Quantization
o_t^{off}
Resource Allocation Algorithm
a_t^{off}
R_t^{ca}
$(s_{t\,(1)}^{off}, a_{t\,(1)}^{off})$
$(s_{t\,(2)}^{off}, a_{t\,(2)}^{off})$
...
$(s_{t\,(M_{off})}^{off}, a_{t\,(M_{off})}^{off})$
Replay Memory
Train the Actor with (41)
$(s_t^{ca}, a_t^{ca}, R_t^{ca}, s_{t+1}^{ca})$
Citric Network
Train the Critic with (39)
$(s_{t\,(1)}^{ca}, a_{t\,(1)}^{ca}, R_{t\,(1)}^{ca}, s_{t+1(1)}^{ca})$
$(s_{t\,(2)}^{ca}, a_{t\,(2)}^{ca}, R_{t\,(2)}^{ca}, s_{t+1(2)}^{ca})$
...
$(s_{t\,(M_{ca})}^{ca}, a_{t\,(M_{ca})}^{ca}, R_{t\,(M_{ca})}^{ca}, s_{t+1(M_{ca})}^{ca})$
Replay Memory

Fig. 2. The framework of the proposed algorithm ACSCTO.



$$\begin{array}{c|cccc|cccc|c}
 & a_1 & a_2 & \cdots & a_K & a_1 & a_2 & \cdots & a_K & \cdots \\
\hline
t_1^t & 0 & d_1^t & \cdots & 0 & 0 & s_1^t & \cdots & 0 & \cdots \\
t_2^t & 0 & 0 & \cdots & d_2^t & 0 & 0 & \cdots & s_2^t & \cdots \\
t_3^t & d_3^t & 0 & \cdots & 0 & s_3^t & 0 & \cdots & 0 & \cdots \\
\vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots & 0 & \cdots \\
t_{N-1}^t & d_{N-1}^t & 0 & \cdots & 0 & s_{N-1}^t & 0 & \cdots & 0 & \cdots \\
t_N^t & 0 & d_N^t & \cdots & 0 & 0 & s_N^t & \cdots & 0 & \cdots
\end{array}$$

Fig. 3. Matrix representation of caching model inputs.

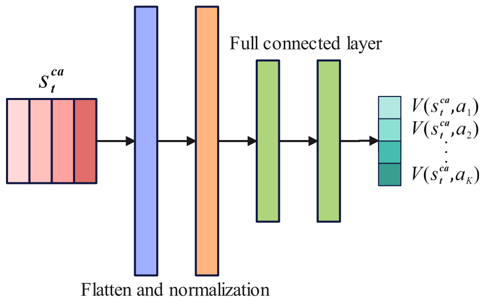


Fig. 4. Critic network architecture.

a softmax transformation, ensuring that their summation equals 1. This result, referred to as Pro^{ca}, denotes the probability distribution of each service to be cached in the ES. To determine the caching decision a_t^{ca}, we conduct non-repetitive sampling based on Pro_t^{ca} until the cumulative storage of the selected services exceeds the maximum caching capacity of the ES.

Regarding the critic network, we have adopted a more efficient approach to augment the learning capacity of the critic, distinguishing it from conventional critic networks. As illustrated in Fig. 4, instead of exclusively generating a single state value, the critic network produces the values for each service in the current time slot. The output can be represented as a vector of size K and the element of the vector is expressed as $V(s_t^{ca}, a_k)$. It shall be noted that, the notation a_k in this context should be clarified as representing service k, rather than denoting an action. With the assistance of the task offloading model, we also map rewards to each service, referred as R_t^{ca}, with details provided in Section VI-B2.

3) Model Learning: We employ backpropagation algorithm and rewards R_t^{ca} derived from the task offloading model to optimize the weights and biases of both the actor and critic networks. During each iteration of training, the actor network, parameterized by ϕ_a, outputs a probability distribution, which dictates the likelihood of caching each service on the ES. Meanwhile, the critic network, parameterized by θ_c, provides a more detailed assessment of state values to better evaluate the efficacy of the selected actions. We denote the critic's output value vector as $V(s_t^{ca}; \theta_c)$, where θ_c is updated by

$$\mathrm{d}\theta_c \leftarrow \mathrm{d}\theta_c + \alpha_c \nabla_{\theta_c} (V(s_t^{ca}; \theta_c) - \hat{V}(s_t^{ca}; \theta_c))^2. \tag{39}$$

The gradient term is the Mean Square Error (MSE) of the predicted value vector with the target value, and

$$\hat{V}(s_t^{ca}; \theta_c) = R_t^{ca} + \gamma^c V(s_t^{ca}; \theta_c). \tag{40}$$

The learning process alternates between policy evaluation and policy improvement. The optimal policy is learned by

$$\begin{aligned}\mathrm{d}\phi_a \leftarrow \mathrm{d}\phi_a - \alpha_a \nabla_{\phi_a} \log(\pi(s_t^{ca}; \phi_a))(\hat{V}(s_t^{ca}; \theta_c) \\ - V(s_t^{ca}; \theta_c)).\end{aligned} \tag{41}$$

Following the experience replay technique, at each time step, we store the newly generated transition $\{s_t^{ca}, a_t^{ca}, R_t^{ca}, s_{t+1}^{ca}\}$ in a replay memory with a limited capacity of M_{ca}. As new samples are generated, they replace the oldest ones in the memory when it reaches full capacity. Then, a batch of samples is randomly selected from this memory for updating the critic and actor networks by (39), (41). By iteratively adjusting the network parameters, both the actor and critic networks progressively enhance their performance and develop the ability to make more informed decisions during the service caching process.

B. Task Offloading Module

1) Model Structure and Model Learning: As depicted in Fig. 5, the task offloading model, parameterized by ϕ_o, takes s_t^{ca} combined with a_t^{ca} as input, denoted as s_t^{off}. Following the flattening and normalization processes described earlier, the

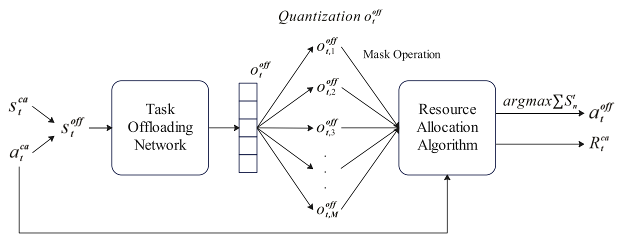


Fig. 5. The schematics of task offloading model.

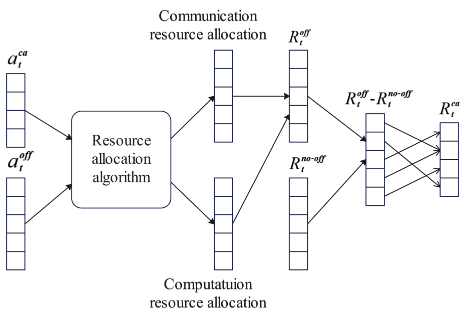

Communication resource allocation
R_t^{off}
a_t^{ca}
R_t^{off}-R_t^{no-off}
R_t^{ca}
a_t^{off}
Resource allocation algorithm
R_t^{no-off}
Computatuion resource allocation

Fig. 6. The calculation process of R_t^{ca}.

input passes through dense layers and then undergoes a sigmoid function, restricting the output o_t^{off} to the value range [0, 1]. The output is continuous and we need binary offloading decisions, so we perform an order-preserving quantization method to obtain M binary operations [31]. Subsequently, we utilize the service caching decision a_t^{ca} to generate a mask vector representing the tasks available for offloading, thereby obtaining M feasible operations. Finally, based on these feasible operations, the user satisfaction value is calculated, and the offloading action a_t^{off} with the highest user satisfaction is recorded.

Within each time frame, the best actions along with their corresponding input states, are stored in an initial memory buffer with a capacity of M_{off}. Then by randomly selecting a batch of samples from M_{off}, we update the network parameters through

$$\begin{aligned} \mathrm{d}\phi_o \leftarrow \mathrm{d}\phi_o - \alpha_o \nabla_{\phi_o}((1-a_t^{off})(1-\log(o_t^{off})) \\ + a_t^{off}\log(o_t^{off})), \end{aligned} \tag{42}$$

where the gradient term represents the average cross-entropy loss between the TOM output o_t^{off} and the recorded offloading action a_t^{off}.

2) Reward of Service Caching Module: According to the aforementioned descriptions, for each a_t^{ca}, the corresponding a_t^{off} can lead to the highest user satisfaction. As depicted in Fig. 6, the Algorithm 3 is then employed to generate the optimal resource allocation strategy given any feasible caching and offloading decision. With this resource allocation strategy, we compute the offloading reward R_t^{off}, representing the user satisfaction for each task, and the no-offloading reward R_t^{no-off}, signifying the user satisfaction for each task when no services are cached. In other words, all the tasks are executed locally. The difference between R_t^{off} and R_t^{no-off} provides the reward for caching each service. By aggregating the rewards for tasks requesting the same service, we determine the caching reward

Algorithm 4: ACSCTO.

```
1:  Initialization:
2:  Caching critic network parameter $\theta_c$;
3:  Caching actor network parameter $\phi_a$;
4:  Offloading network parameter $\phi_o$;
5:  Replay memory $Mem_{ca}$, $Mem_{off}$ with size $M_{ca}$ and $M_{off}$;
6:  for $t = 1, \ldots, T$ do
7:    Caching actor network generate $a_t^{ca}$ as the caching decision in time slot $t$;
8:    Delivery $a_t^{ca}$ to offloading network, generate offloading decision $a_t^{off}$ in time slot $t$;
9:    Store the tuple $(s_t^{off}, a_t^{off})$ in replay memory $Mem_{off}$;
10:   Obtain resource allocation decision using Algorithm 3;
11:   Perform generate actions, obtain reward $R_t^{ca}$ and next state $s_{t+1}^{ca}$;
12:   Store the tuple $(s_t^{ca}, a_t^{ca}, R_t^{ca}, s_{t+1}^{ca})$ in replay memory $Mem_{ca}$;
13:   if $Mem_{ca}$ and $Mem_{off}$ are full then
14:     Sample a batch from $Mem_{ca}$ and update the critic and actor network according to (39) and (41), respectively;
15:     Sample a batch from $Mem_{off}$ and update the offloading network according to (42);
16:   end if
17:   $t \longleftarrow t+1$;
18: end for
```

for each service, denoted as R_t^{ca}, which can guide the service caching module training.

Algorithm 4 offers a comprehensive overview of the model update process and iterative optimization procedure of the proposed ACSCTO algorithm. To determine the optimal scheduling decisions for each time slot, we iteratively update the service caching module and task offloading module and utilize the alogrithm 3 to determine the best resource allocation strategy. This decomposition of the problem allows the model to dynamically adapt to the environment and address the specific requirements of complex users and services more effectively.

VII. Performance Evaluation

In this section, we conduct a comprehensive performance evaluation of the proposed optimization method, employing diverse metrics to assess its efficacy.

A. Experimental Setups

The depicted cloud-edge framework in this paper is meticulously designed and rigorously stimulated. Within this framework, a total of 10 services are presented, and each service requires storage resources that follow a uniform distribution ranging from 1 GB to 4 GB. The energy consumption for caching each service per application adheres to a uniform distribution between 2 to 5. The central BS with an edge server is located

TABLE III
NOTATIONS AND DESCRIPTIONS

Notation	Value	Notation	Value
N	30	K	10
d_n^t	[1,6]	s_n^t	[1,5]
$l_n^{max,t}$	[0.7,3]	h_k	[1,4]
γ_k	[2,5]	W	30
F_e	40	C	10

within a 100x100 unit area, and surrounded by uniformly distributed UDs. The computation tasks involve input data sizes ranging uniformly from 1 MB to 6 MB, CPU requirements spanning from 1 GHz to 5 GHz, and task deadlines uniformly ranging from 0.7 to 3 seconds. The ES features a storage capacity of 10 GB, a computation capacity of 40 GHz, and a network capacity totaling 30 GHz. In relation to the service request for each task, we employ a request transition model [32], [33], [34]. Specifically, the transition probability of requesting any service k in current time slot, given the previous request for service i, adheres to a uniform distribution. In addition, the primary simulation parameters are shown in Table III.

B. Evaluation Metrics

To assess the effectiveness of the proposed ACSCTO algorithm, we consider key metrics in the edge computing scenario.

1) *Average user satisfaction:* The average satisfaction level of users, calculated by (9).
2) *Average response time:* The average duration to respond to user requests.
3) *Tasks completion rate:* The ratio of tasks completed within the specified deadlines to the total number of tasks.
4) *Inter-user fairness:* The fairness degree among users, indicated by the variance of task completion times.

Other important metrics include: **Average overtime**: the mean excess time tasks surpass their specified deadlines; **Energy consumption**: The average energy consumption at ES during each simulation period.; **Average service downloading delay** and **Average task offloading number**.

C. Baseline Algorithms

In our performance evaluation, we compare our proposed algorithm against the following baseline algorithms:

AC with even allocation (AC-ea): This approach adopts the same caching and offloading policies as presented in the ACSCTO algorithm, while uniformly allocating communication [35] and computing resources (i.e., bandwidth and CPU frequency) to the offloaded tasks [36].

AC with single value (AC-sv): Compared to AC-ea, this approach utilizes the Advantage Actor-Critic framework to estimate a single state value [37], rather than estimate multiple values as described in Section VI-A2.

AC without decomposition (AC-wd): This approach utilizes the Advantage Actor-Critic framework to estimate the state function, and directly determines both caching and offloading action policies to improve overall system performance [38].

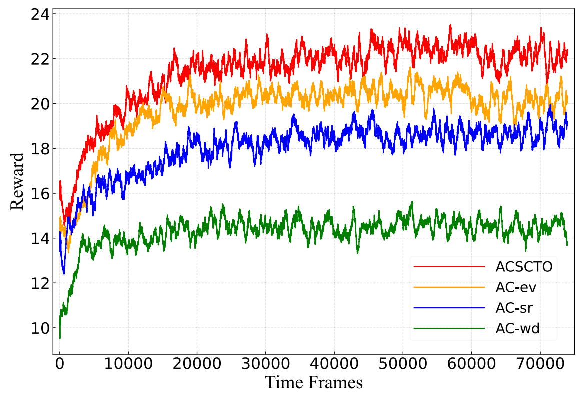


Fig. 7. The convergence performance.

Popularity caching (PC): This approach makes caching decisions according to their popularity, and the popularity of one service is defined based on the frequency of service requests [39].

Local Computing (LC): All the tasks are executed locally.

D. Convergence Performance

In Fig. 7, we compare the convergence speed and final reward performance of four Actor-Critic-based algorithms: ACSCTO, AC-ea, AC-sv, and AC-wd, within a cache-assisted edge computing environment comprising 30 UDs. The reward at each time slot is denoted as $\sum_{k\in\mathcal{K}} R_{t,k}^c a$. All curves are plotted using a moving average with a window size of 500.

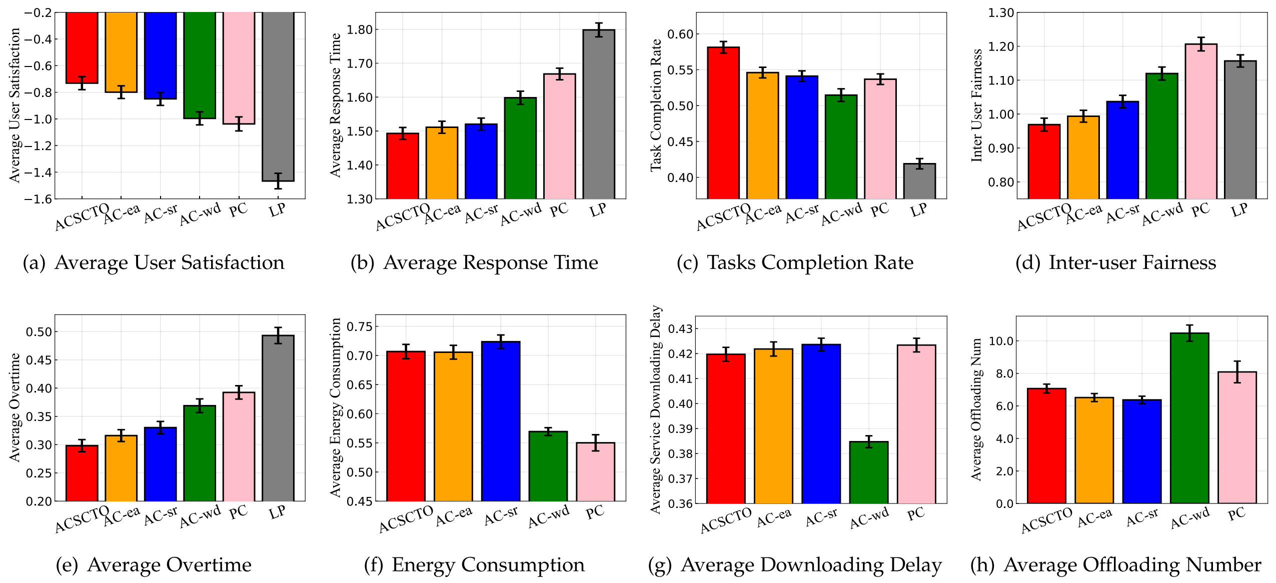
During the training process, the reward performance of various exploration methods exhibited dynamic characteristics. These algorithms demonstrated rapid convergence in the initial stage ($t < 5000$), followed by a gradual slowdown, and then approach convergence at approximately 20,000 time frames. There are some fluctuations, due to the stochastic nature of the MEC environment.

Although these algorithms exhibited similar convergence speeds, significant differences were observed in their final rewards. For instance, due to the multiple value simulation approach of AC-ea, its average reward exceeded that of AC-sv, validating the efficacy of our proposed approach. Furthermore, ACSCTO exhibited the highest final reward performance, compared to other three algorithms. The simulation result highlights the advantage of the ACSCTO algorithm in optimizing edge resource allocation and task scheduling.

E. Evaluation With Baseline Algorithms

A comprehensive evaluation of all algorithms was conducted, and the evaluation metrics include average user satisfaction, average response time, task completion rate, and inter-user fairness. Fig. 8 presents the simulation results obtained from simulating 10000 time slots on the test data set.

Fig. 8(a) illustrates the average user satisfaction across all algorithms when the number of UDs is set to 30. Given the stringently configured MEC environment, the average user satisfaction for all algorithms is negative. Among the baseline algorithms, AC-ev and AC-sr exhibit higher user satisfaction



(a) Average User Satisfaction (b) Average Response Time (c) Tasks Completion Rate (d) Inter-user Fairness

(e) Average Overtime (f) Energy Consumption (g) Average Downloading Delay (h) Average Offloading Number

Fig. 8. Comparison of ACSCTO model with baselines.

compared to other baseline algorithms. Notably, the user satisfaction and task completion rate of the AC-ev model surpasses that of AC-sr, attributed to the former's precise evaluation of each service's value. Fig. 8(g) and (h) reveal that compared to AC-sr, AC-ev approach allocates less time to service updates, but demonstrates a higher rate of task offloading to the edge server. It indicates that AC-ev possesses the capability to optimize service caching decisions, thereby resulting in reduced service downloading delay, improved task offloading decisions, and heightened user satisfaction.

The user satisfaction achieved by ACSCTO algorithm surpasses that of the top-performing baseline algorithm, AC-ev, since ACSCTO can inherit all the previously discussed strengths of AC-ev algorithm. Furthermore, the integration of a convex optimizer for resource allocation in ACSCTO enhances the overall algorithm's upper limit. Hence, ACSCTO demonstrates swift adaptation to task workload behaviors and offers greater flexibility in resource allocation. When faced with urgent offloading tasks, ACSCTO model assigns more transmission and computing resources to prevent delays surpassing the time limit, thereby averting penalty due to delayed task completion.

Consequently, as shown in Fig. 8(b), (c), (d), and (e), the ACSCTO strategy yields the shortest average task response times, lowest average overtime, highest task completion rate and fairness among all scheduling policies. Hence, ACSCTO can efficiently address the stringent latency constraints of time-critical applications, such as healthcare monitoring, autonomous driving, and real-time gaming, as exemplified in the previous section. Moreover, the energy consumption of the ACSCTO algorithm in the ES remains comparable to that of the best baseline algorithm, AC-ev. The ACSCTO algorithm can achieve prior utilization of the limited resources available in the ES.

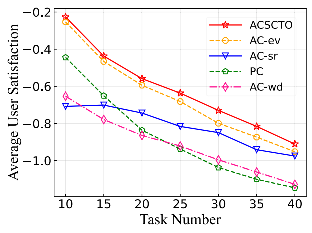
In Fig. 8, the green bars represent the AC-wd algorithm, which, unlike the ACSCTO algorithm, does not separate the caching module from the offloading module. Instead, it utilizes the A2C framework to identify the optimal caching and offloading actions. However, it fails to comprehend the correspondence between services and tasks. Due to the absence of task selection aligned with the caching service will result in a significant reduction in rewards, as depicted in Fig. 8(h), the task scheduling strategy of the AC-wo method is more aggressive, resulting in the offloading of the highest number of tasks to the resource-constrained ES. Consequently, its performance is the poorest among all AC-based algorithms.

The PC algorithm is a heuristic method designed to place frequently requested services on the ES and prioritize offloading more urgent tasks to the ES. However, heuristic algorithms inherently lack generalizability and quick adaptability, making them prone to drawing erroneous conclusions. The PC method not only demonstrated the second-worst performance, after the LP method, in metrics such as average user satisfaction, average response time, and task completion rate, but also exhibited the most significant fluctuations in average offloading number, as shown in Fig. 8(h). Moreover, Fig. 8(d) reveals that it has the worst inter-user fairness.

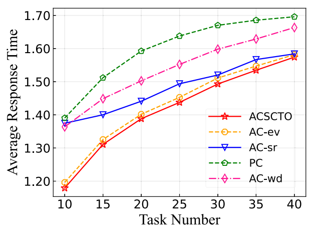
F. Comparison Under Different User Device Number and Cache Size

In the following analysis, we compare various performance metrics across diverse network configurations. These metrics include average user satisfaction, average task response time, task completion rate, and inter-user fairness. Specifically, we investigate these metrics with different number of user devices and ES cache capacities. Fig. 9(a)–(d) illustrate the variations in user devices from 10 to 40. Fig. 10(a)–(d) showcase modifications in ES storage capacity from 6 to 12 with 20 user devices.

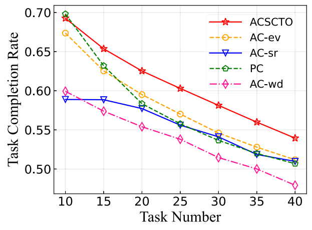
Fig. 9 illustrates that algorithms such as ACSCTO and AC-ea, which accurately estimate the value of each service, show more pronounced advantages. As the number of user devices increases



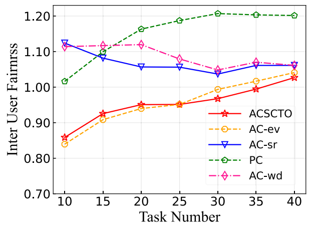
(a) Average User Satisfaction



(b) Average Response Time

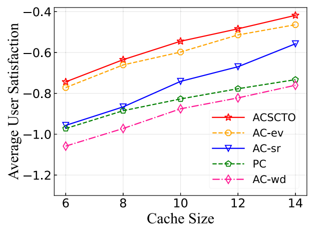


(c) Tasks Completion Rate

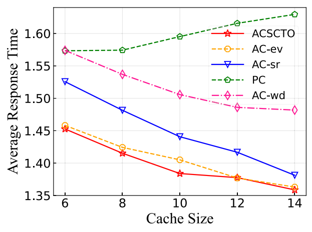


(d) Inter-user Fairness

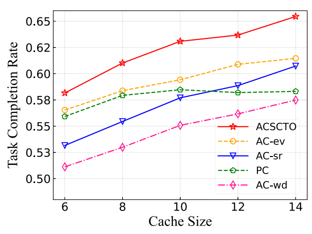
Fig. 9. Comparison of ACSCTO model with baselines against different UDs.



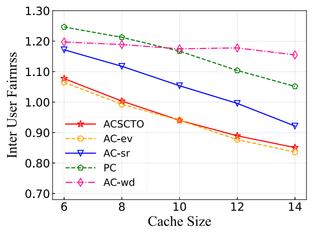
(a) Average User Satisfaction



(b) Average Response Time



(c) Tasks Completion Rate



(d) Inter-user Fairness

Fig. 10. Comparison of ACSCTO model with baselines against different cache sizes.

from 10 to 40, both the average user satisfaction and task completion rate decrease, while task response time increases. In other words, the effectiveness of each algorithm deteriorates. This phenomenon arises because, with unchanged transmission and computing resources at ES and more offloaded tasks, the competition among resources on the ES intensifies. As a result, the resources available to each task decrease, diluting the benefits of offloading to the ES. It is evident that the differences between ACSCTO, AC-ea, and AC-sr diminish. This is because, as the number of users increases, the number of tasks corresponding to each service also increases, and the benefits of caching each service tend to become homogeneous. At this point, the AC-ea algorithm degrades to the AC-sr algorithm, making it challenging to distinguish between various algorithms based on offloading and resource allocation strategies.

TABLE IV
NUMBER OF ITERATIONS REQUIRED FOR CONVERGENCE

	Outer Loop			Inner Loop		
User Num	Max	Min	Avg	Max	Min	Avg
10	36	27	33.52	41	11	24.69
20	36	29	33.64	40	9	25.18
30	37	27	33.41	41	10	26.01
40	36	29	33.40	41	10	26.00
Delay Offset	Max	Min	Avg	Max	Min	Avg
-0.2	36	28	33.04	40	10	25.12
-0.1	36	30	33.41	42	6	24.85
0	36	29	33.64	40	9	25.18
+0.1	37	30	33.97	42	9	25.37
+0.2	37	30	34.07	42	9	25.68

When the number of user devices remains 20 and the ES cache capacity decreases, the proposed ACSCTO strategy continues to outperform others. As more services are allowed to be cached on the ES, the hit rate will increase, allowing more tasks to be executed in the ES. By choosing suitable tasks for offloading, the response time for user requests is markedly decreased, resulting in a notable enhancement in user satisfaction. Notably, the PC approach is the only one that becomes worse as the cache capacity of the edge server increases. As shown in Fig. 10(b) and (c), the task completion rate of the PC method grows slowly and gradually plateaus, while the task response time increases. This indicates that heuristic methods lack adaptability and flexibility, failing to effectively adjust strategies to accommodate changing environments and exhibiting limitations in handling dynamic changes and resource scaling.

G. Convergence of the BRA Algorithm

We conducted an analysis of the convergence behavior of the BRA algorithm. Table IV presents the statistics on the maximum, minimum, and average iteration counts of the inner and outer loops of the BRA algorithm, when the number of UDs increases from 10 to 40 and the task delay constraint bias varies from -0.2 to +0.2, with 20 UDs. From Table IV, it can be observed that regardless of variations in the number of users, the differences between the minimum and maximum iteration counts, as well as the fluctuations in the average iteration counts for both inner and outer loops, are minimal. This is primarily attributed to the adoption of vectorized techniques, effectively optimizing the execution efficiency of the algorithm.

As the task delay constraint gradually increases, reducing inter-task competition and rendering resource allocation more flexible, there is a slight increase in the iteration counts of the inner loop, albeit negligible. Overall, the results presented in Table IV demonstrate that the BRA algorithm consistently converges under different conditions and maintains good performance. This further validates the applicability and practicality of the BRA algorithm in edge computing environments.

VIII. Conclusion

This paper considered a joint optimization problem in a dynamic multi-user, multi-service MEC network, with the goal of maximizing average user satisfaction. We put forward a two-stage solution to the modeled MINLP problem. First, we model the service and task scheduling problems as the Markov Decision Process and derive policies through the A2C network. Next, we use a convex optimizer based on the Lagrangian duality method to allocate bandwidth and computing resources, given caching and offloading policies. The proposed ACSCTO approach demonstrates significant enhancements in average user satisfaction, average task response time, task completion rate, and inter-user fairness when compared to several baseline approaches. For the future work, we plan to explore the joint optimization problem in cloud-edge collaborative scenarios involving multiple edge servers.

References

[1] C. Tang, H. Wu, R. Li, C. Zhu, and J. J. P. C. Rodrigues, "A systematic exploration of edge computing-enabled metaverse," *IEEE Netw.*, vol. 37, no. 6, pp. 10–17, Nov. 2023.

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[3] Z. Peng, G. Wang, W. Nong, Y. Qiu, and S. Huang, "Task offloading in multiple-services mobile edge computing: A deep reinforcement learning algorithm," *Comput. Commun.*, vol. 202, pp. 1–12, 2023.

[4] D. Triyanto, I. W. Mustika Widyawan, and P. Pavarangkoon, "Fairness-aware computation offloading for mobile edge computing with energy harvesting," *IEEE Access*, vol. 13, pp. 55 345–55 357, 2025.

[5] X. Li, X. Chen, and G. Li, "Fairness-aware task offloading and load balancing with delay constraints for power Internet of Things," *Ad Hoc Netw.*, vol. 153, 2024, Art. no. 103333.

[6] C. Ling, W. Zhang, H. He, R. Yadav, J. Wang, and D. Wang, "QoS and fairness oriented dynamic computation offloading in the internet of vehicles based on estimate time of arrival," *IEEE Trans. Veh. Technol.*, vol. 73, no. 7, pp. 10 554–10 571, Jul. 2024.

[7] C. Tang, Y. Zhao, and H. Wu, "Lyapunov-guided optimal service placement in vehicular edge computing," *China Commun.*, vol. 20, no. 3, pp. 201–217, 2023.

[8] T. T. Vu, D. T. Hoang, K. T. Phan, D. N. Nguyen, and E. Dutkiewicz, "Energy-based proportional fairness for task offloading and resource allocation in edge computing," in *Proc. 2022 IEEE Int. Conf. Commun.*, 2022, pp. 1912–1917.

[9] A. Xiao, S. Wu, Y. Ou, N. Chen, C. Jiang, and W. Zhang, "QoE-fairness-aware bandwidth allocation design for MEC-assisted ABR video transmission," *IEEE Trans. Netw. Service Manag.*, vol. 22, no. 1, pp. 499–515, Feb. 2025.

[10] K. Peng, C. Ling, B. Zhao, and V. C. M. Leung, "A fairness-aware task offloading method in edge-enabled IIoT with multi-constraints using AGE-MOEA and weighted MMF," *Concurr. Comput. Pract. Exp.*, vol. 36, no. 10, 2024, Art. no. e7858.

[11] X. Bi, X. Sun, Z. Lyu, B. Zhang, and X. Wei, "A back adjustment based dependent task offloading scheduling algorithm with fairness constraints in VEC networks," *Comput. Netw.*, vol. 223, 2023, Art. no. 109552.

[12] R. Karmakar, G. Kaddoum, and O. Akhrif, "A novel federated learning-based smart power and 3D trajectory control for fairness optimization in secure UAV-assisted MEC services," *IEEE Trans. Mob. Comput.*, vol. 23, no. 5, pp. 4832–4848, May 2024.

[13] S. Wang, X. Song, T. Song, and Y. Yang, "Fairness-aware computation offloading with trajectory optimization and phase-shift design in RIS-assisted multi-UAV MEC network," *IEEE Internet Things J.*, vol. 11, no. 11, pp. 20 547–20 561, Jun. 2024.

[14] Z. Fu, L. Shi, Y. Ye, Y. Zhang, and G. Zheng, "Computation EE fairness for a UAV-enabled wireless powered MEC network with hybrid passive and active transmissions," *IEEE Internet Things J.*, vol. 11, no. 11, pp. 20 152–20 164, Jun. 2024.

[15] Z. Wan, W. Jiang, J. Nie, D. Niyato, C. Pan, and Z. Xiong, "Min-max fairness based joint optimal design for IRS-assisted MEC systems," *IEEE Trans. Veh. Technol.*, vol. 73, no. 8, pp. 11 949–11 963, Aug. 2024.

[16] X. Dai et al., "Task offloading for cloud-assisted fog computing with dynamic service caching in enterprise management systems," *IEEE Trans. Ind. Informat.*, vol. 19, no. 1, pp. 662–672, Jan. 2023.

[17] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *Proc. IEEE Conf. Comput. Commun.*, Honolulu, HI, USA, Apr. 16–19, 2018, pp. 207–215.

[18] C. Tang, C. Zhu, H. Wu, Q. Li, and J. J. P. C. Rodrigues, "Toward response time minimization considering energy consumption in caching-assisted vehicular edge computing," *IEEE Internet Things J.*, vol. 9, no. 7, pp. 5051–5064, Apr. 2022.

[19] W. Fan et al., "Collaborative service placement, task scheduling, and resource allocation for task offloading with edge-cloud cooperation," *IEEE Trans. Mobile Comput.*, vol. 23, no. 1, pp. 238–256, Jan. 2024.

[20] Z. Wang, M. Li, L. Zhao, H. Zhou, and N. Wang, "A3C-based computation offloading and service caching in cloud-edge computing networks," in *Proc. IEEE Conf. Comput. Commun. Workshops*, 2022, pp. 1–2.

[21] H. Yan, X. Xu, F. Dai, L. Qi, X. Zhang, and W. Dou, "Service caching for meteorological emergency decision-making in cloud-edge computing," in *Proc. 2022 IEEE Int. Conf. Web Serv.*, 2022, pp. 120–128.

[22] S. Tang et al., "Fairness-efficiency scheduling for pay-as-you-go shared caching systems with long-term fairness guarantees," *IEEE Trans. Serv. Comput.*, vol. 17, no. 5, pp. 2417–2431, Sep./OCt. 2024.

[23] L. Chen, S. Zheng, Y. Wu, H. Dai, and J. Wu, "Resource and fairness-aware digital twin service caching and request routing with edge collaboration," *IEEE Wirel. Commun. Lett.*, vol. 12, no. 11, pp. 1881–1885, Nov. 2023.

[24] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Service placement and request routing in MEC networks with storage, computation, and communication constraints," *IEEE/ACM Trans. Netw.*, vol. 28, no. 3, pp. 1047–1060, Jun. 2020.

[25] Y. Li et al., "Cooperative service placement and scheduling in edge clouds: A deadline-driven approach," *IEEE Trans. Mobile Comput.*, vol. 21, no. 10, pp. 3519–3535, Oct. 2022.

[26] H. Zhou, K. Jiang, X. Liu, X. Li, and V. C. M. Leung, "Deep reinforcement learning for energy-efficient computation offloading in mobile-edge computing," *IEEE Internet Things J.*, vol. 9, no. 2, pp. 1517–1530, Jan. 2022.

[27] M. Poposka, Z. Hadzi-Velkov, and T. Shuminoski, "Proportional fairness in wireless powered mobile edge computing networks," in *Proc. 30th Int. Conf. Syst. Signals Image Process.*, 2023, pp. 1–5.

[28] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan, "Rate control for communication networks:shadow prices, proportional fairness and stability," *J. Oper. Res. Soc.*, vol. 49, no. 3, pp. 237–252, 1998.

[29] S. Wang, S. Bi, and Y.-J. A. Zhang, "Deep reinforcement learning with communication transformer for adaptive live streaming in wireless edge networks," *IEEE J. Sel. Areas Commun.*, vol. 40, no. 1, pp. 308–322, Jan. 2022.

[30] D. Pathak, P. Krähenbühl, and T. Darrell, "Constrained convolutional neural networks for weakly supervised segmentation," in *Proc. 2015 IEEE Int. Conf. Comput. Vis.*, Santiago, Chile, Dec. 7–13, 2015, pp. 1796–1804.

[31] L. Huang, S. Bi, and Y.-J. A. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Trans. Mobile Comput.*, vol. 19, no. 11, pp. 2581–2593, Nov. 2020.

[32] Y. Sun, Y. Cui, and H. Liu, "Joint pushing and caching for bandwidth utilization maximization in wireless networks," *IEEE Trans. Commun.*, vol. 67, no. 1, pp. 391–404, Jan. 2019.

[33] Y. Qian, R. Wang, J. Wu, B. Tan, and H. Ren, "Reinforcement learning-based optimal computing and caching in mobile edge network," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 10, pp. 2343–2355, Oct. 2020.

[34] Z. Chen, W. Yi, A. S. Alam, and A. Nallanathan, "Dynamic task software caching-assisted computation offloading for multi-access edge computing," *IEEE Trans. Commun.*, vol. 70, no. 10, pp. 6950–6965, Oct. 2022.

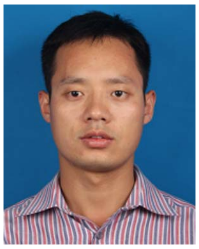
[35] J. Zhou and X. Zhang, "Fairness-aware task offloading and resource allocation in cooperative mobile-edge computing," *IEEE Internet Things J.*, vol. 9, no. 5, pp. 3812–3824, Mar. 2022.

[36] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 1, pp. 856–868, Jan. 2019.

[37] P. Choppara and S. Mangalampalli, "Reliability and trust aware task scheduler for cloud-fog computing using advantage actor critic (A2C) algorithm," *IEEE Access*, vol. 12, pp. 102 126–102 145, 2024.

[38] J. Lu et al., "A2C-DRL: Dynamic scheduling for stochastic edge-cloud environments using A2C and deep reinforcement learning," *IEEE Internet Things J.*, vol. 11, no. 9, pp. 16 915–16 927, May 2024.

[39] X. Wei, J. Liu, Y. Wang, C. Tang, and Y. Hu, "Wireless edge caching based on content similarity in dynamic environments," *J. Syst. Archit.*, vol. 115, 2021, Art. no. 102000.



Chaogang Tang (Member, IEEE) received the BS degree from the Nanjing University of Aeronautics and Astronautics, Nanjing, China, and the PhD degree from the School of Information Science and Technology, University of Science and Technology of China, Hefei, China, and the Department of Computer Science, City University of Hong Kong, under a joint PhD Program, in 2012. He is now with the China University of Mining and Technology. His research interests include mobile cloud computing, fog computing, Internet of Things, Big Data.

Yao Ding received the BE degree from the Nanjing Agricultural University (NAU), Nanjing, China, in 2020. She is currently working toward the postgraduate degree with the China University of Mining and Technology, CUMT. Her main research topics include edge computing, edge intelligence, Internet of Things and machine learning.

Shuo Xiao received the PhD degree in traffic information engineering and control from Beijing Jiaotong University. He is currently a professor of Computer Science and Technology, China University of Mining and Technology. His research interests include intelligent information processing, artificial intelligence and pattern recognition, machine learning.

Zhenzhen Huang received the PhD degree in computer science and technology from the China University of Mining and Technology. She is now an associate professor. Her research interests include recommendation and artificial intelligence technologies.

Huaming Wu (Senior Member, IEEE) received the BE and MS degrees in electrical engineering from the Harbin Institute of Technology, China in 2009 and 2011, respectively and the PhD degree with the highest honor in computer science, Freie Universität Berlin, Germany in 2015. He is currently a professor with the Center for Applied Mathematics, Tianjin University. His research interests include model-based evaluation, wireless and mobile network systems, mobile cloud computing and deep learning.