

Article

Joint Task Offloading and Resource Allocation with Data Caching in UAV-Aided Mobile Edge Computing Networks for Latency-Sensitive Applications

Tanmay Baidya and Sangman Moh *

Department of Computer Engineering, Chosun University, 309 Pilmun-daero, Dong-gu, Gwangju 61452, Republic of Korea

* Correspondence: smmoh@chosun.ac.kr; Tel.: +82-62-230-6032

Abstract

The rapid growth of computing-intensive and latency-sensitive applications, including augmented reality, virtual reality, and self-driving systems, has increased the demand for low-latency and energy-efficient processing solutions. Mobile edge computing (MEC) has evolved as a transformative paradigm by relocating computation to the network edge, closer to end users. Unmanned aerial vehicles (UAVs) further strengthen MEC by offering flexible deployment, mobility, and reliable line-of-sight communication, making them suitable for temporary high-demand scenarios. Moreover, such latency-sensitive applications often generate numerous repetitive tasks and, thus, storing the results of these tasks can reduce both communication overhead and computational workload. However, jointly addressing the caching of task-results alongside offloading and resource allocation decisions in UAV-aided MEC networks remains a non-trivial challenge. In this study, an integrated task offloading and resource allocation with data caching (JORC) framework is proposed to address these challenges. The offloading and resource allocation problems are formulated as a Markov decision process and solved using the soft actor-critic reinforcement learning algorithm. In addition, dynamic and adaptive caching manages limited storage and reduces redundant computations by using a hybrid strategy that integrates the least-frequently used and least-recently used policies to reduce computational redundancy. Simulation results confirm that the proposed JORC framework substantially reduces latency, energy consumption, and overall system cost, while increasing the successful task completion ratio compared to existing baseline approaches.

Keywords: caching; deep reinforcement learning; mobile edge computing; offloading; resource allocation; soft actor-critic; unmanned aerial vehicle

Academic Editors: Wei Yi, Chase Wu and Guoxin Zhang

Received: 11 June 2026

Revised: 24 July 2026

Accepted: 30 July 2026

Published: 5 August 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

The rapid advancement of latency-sensitive and computation-intensive applications, including augmented reality (AR), virtual reality (VR), autonomous vehicles, and Internet of Things (IoT) technologies, is transforming digital ecosystems. These technologies demand real-time processing with extremely low response delays to maintain uninterrupted system performance and a high-quality user experience [1]. For instance, in AR/VR applications, the real-time rendering of high-resolution graphics requires rapid data transmission and low-latency computing to avoid motion sickness and visual lag [2]. Autonomous

vehicles rely on continuous environmental perception, sensor fusion, and rapid decision making to navigate safely, necessitating millisecond-level inferences to prevent accidents [3]. Similarly, IoT networks generate vast amounts of data from distributed edge devices, thereby requiring efficient real-time processing for services including intelligent urban infrastructure automated industrial systems, and remote medical surveillance [4].

Traditional cloud-computing architectures struggle to meet these stringent requirements owing to high transmission delays, network congestion, and excessive energy consumption associated with long-distance data processing. Consequently, mobile edge computing (MEC) is regarded as a fundamental paradigm that places computing capabilities near end devices, which substantially decreases communication delay and improves energy utilization efficiency [3]. MEC enables edge nodes, such as access points, base stations, and small-scale datacenters, to compute tasks locally rather than offloading them to distant cloud servers. By minimizing data-transmission delays and network congestion, MEC provides low-latency computation, improves bandwidth utilization, and reduces power consumption [5]. MEC also facilitates AI-driven edge intelligence, enabling real-time inference, predictive analytics, and adaptive resource management at the network edges. Despite these advantages, MEC faces critical challenges, especially under highly dynamic scenarios characterized by varying workload requirements and changing network states. Effective management of computation offloading and resource distribution is necessary to maintain system responsiveness and energy efficiency. These challenges become even more complex in large-scale or infrastructure-sparse scenarios [6].

To enhance the flexibility and scalability of MEC networks, unmanned aerial vehicles (UAVs) are increasingly being adopted as mobile edge nodes to provide on-demand computational resources and network coverage in dynamic environments. UAVs are particularly advantageous because of their adaptive mobility, rapid deployment flexibility, and line-of-sight (LoS) transmission characteristics, which make them suitable for temporary high-traffic scenarios, disaster recovery operations, and remote area connectivity [7]. For instance, in a large-scale public event, such as a stadium championship match, network congestion due to thousands of simultaneous users can degrade service quality. Deploying UAV-assisted MEC nodes in such an environment can dynamically offload computational tasks, enhance network capacity, and reduce latency for the real-time services, such as AR/VR overlays, live video streaming, and real-time analytics [8]. Moreover, UAVs can support emergency communication in disaster-stricken regions in which conventional ground-based infrastructure becomes unavailable or damaged [9].

Within UAV-assisted MEC architectures, task offloading serves as a fundamental mechanism for improving quality of service (QoS) through the transfer of computational workloads from user equipment toward edge computing nodes for processing [8]. However, UAVs have limited processing capabilities owing to hardware constraints, and the blind offloading of all tasks can result in network congestion, resource depletion, and increased processing delays [10]. Additionally, available communication resources, including channel bandwidth and transmission energy, remain constrained, resulting in competition among multiple devices attempting to access the computational resources of UAVs [11]. Without efficient computation offloading and resource management strategies, the system may experience inefficient resource utilization, prolonged communication delays, and excessive energy usage [12]. Therefore, an optimized offloading strategy is essential to determine which tasks are processed locally, which need to be offloaded, and how to distribute available resources efficiently to balance the computational load and minimize system overhead.

Beyond computation and communication, caching can play a pivotal role in enhancing MEC performance. In this study, caching does not refer to conventional content caching, such as storing videos, files, or service programs. Instead, we consider task-result

caching, where the output of a previously completed deterministic computation task is stored and reused when the same task with identical input is requested again. This approach is particularly meaningful for latency-sensitive MEC applications that generate repetitive computation requests. For example, in AR applications, rendering the same virtual scene or object from a similar viewpoint often requires repeating an identical rendering computation [13,14]. In IoT sensor networks, periodic sensor readings under similar conditions can trigger repeated inference computations, and in autonomous systems, similar sensor inputs, such as recurring traffic scenes, can lead to repeated image-classification computations [14,15]. In such cases, caching the results of frequently requested tasks can significantly reduce both transmission and processing overhead. If a result is already cached, it can be delivered instantly to the user, bypassing the need for recomputation. This not only improves response time but also lowers energy consumption [15]. However, due to limited storage capacity at UAVs and MEC nodes, cache management approaches should be optimally configured to improve the cache hit ratio while operating within resource constraints [12].

This study investigates a UAV-aided MEC network, in which several UAVs integrated with edge computing units and cache storages cooperate with a base station (BS) with higher computing and caching capabilities, as shown in Figure 1. The network serves mobile user devices (UDs) on the ground, including smartphones, AR/VR devices, and IoT sensors that generate computational tasks. Each UAV operates at a fixed altitude and provides computing services within its dedicated coverage area where the user density is high. UD can execute tasks locally or offload them to their assigned UAV, which then determines whether to compute the task locally on UAV or further offload it to the BS for assistance, considering its limited computational capacity.

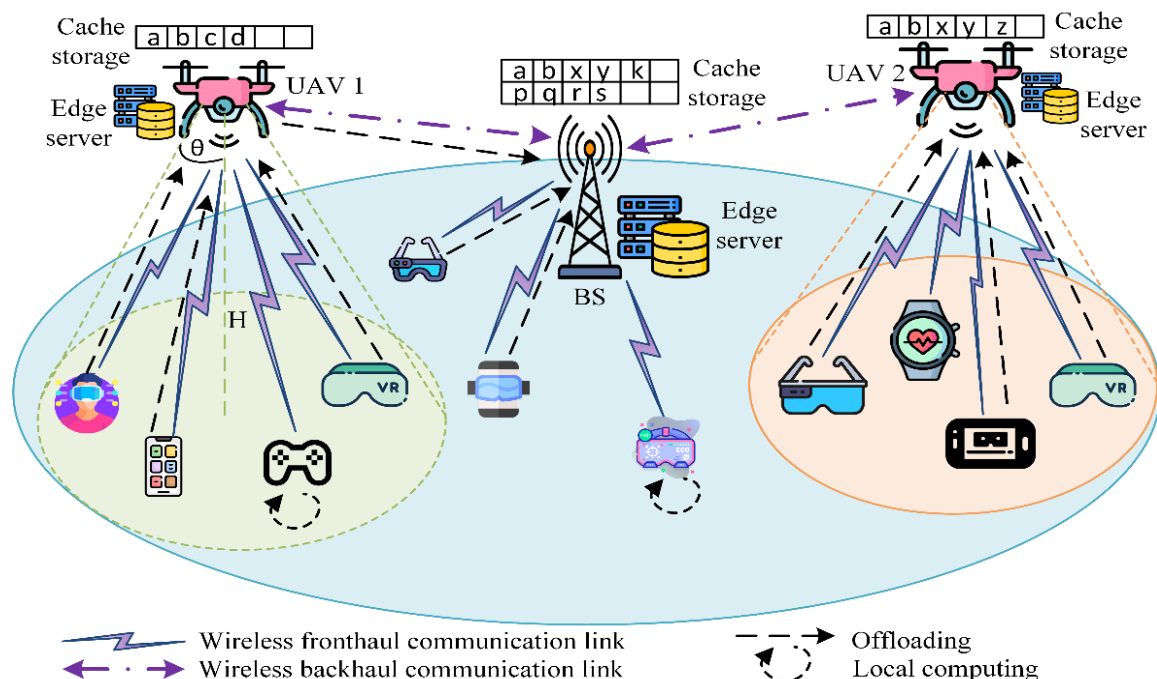


Figure 1. The system model of joint task offloading and resource allocation with data caching in UAV-aided MEC networks.

To reduce latency and energy consumption, tasks must be efficiently offloaded, and resources must be allocated properly. UD allocate transmission power for offloading, whereas UAVs and BS allocate computing resources for processing. To further reduce computational redundancy, completed task-results can be stored in UAV or BS caches

such that repetitive requests can be served directly without reprocessing. This significantly reduces communication and computation overhead and improves system efficiency. However, due to constrained cache storage availability, not all processed tasks can be stored, necessitating an optimized caching policy. Additionally, all tasks have latency constraints; therefore, an effective approach for offloading, resource management, and cache coordination is necessary to guarantee successful task completion.

Although task offloading, resource allocation, and caching have each been widely investigated in UAV-aided MEC networks, most existing studies address only one or two of these components at a time. To the best of our knowledge, among the limited studies that attempt to jointly optimize all three, none combine a UAV-BS collaborative architecture with an entropy-regularized soft actor–critic (SAC)-based deep reinforcement learning (DRL) method for mixed binary-continuous decision making together with a reusable task-result caching mechanism. Existing joint solutions rely on non-learning optimization methods with limited adaptability to dynamic network states, employ DRL algorithms (e.g., DQN, DDQN) that face exploration or scalability limitations in continuous, high-dimensional action spaces, or focus on caching pre-deployed content or services rather than the results of completed computations.

To address this gap, we propose a joint task offloading and resource allocation with data caching (JORC) framework. The considered optimization model is represented as a mixed-integer nonlinear optimization problem owing to the presence of both binary and continuous-valued decision parameters. To solve this problem, we modeled the problem as a Markov decision process (MDP) and employed a DRL method based on the SAC algorithm for optimizing task offloading and resource allocation, along with a hybrid least-frequently used and least-recently used (LFU-LRU) caching mechanism for adaptive data caching. The JORC scheme enables the system to learn adaptive policies that balance latency, energy consumption, and resource utilization under dynamic network conditions.

The primary contributions of this study are summarized below:

- **UAV-BS collaborative architecture:** We designed a UAV-aided MEC framework in which multiple UAVs, each equipped with edge computing and caching capabilities, collaborate with a terrestrial BS to jointly support delay-sensitive and computation-demanding applications, rather than relying on UAV-only or BS-only processing.
- **Joint task offloading and resource allocation with data caching (JORC) scheme:** We developed a JORC framework that integrates SAC-based offloading and resource allocation decisions with a hybrid LFU-LRU task-result caching mechanism within a UAV-BS collaborative architecture. The joint problem is formulated as an MDP and solved via an entropy-regularized, continuous-control DRL approach, enabling adaptive policies that reduce redundant computation while balancing latency, energy, and resource utilization.
- **Adaptive caching mechanism:** We designed a dynamic cache replacement mechanism that jointly considers the access frequency and recency of cached task-result entries. This enables the system to retain computation results with higher reuse probability while adapting to changing request patterns and limited UAV/BS cache capacity.
- **Extensive simulation and performance evaluation:** We conducted comprehensive experiments demonstrating that JORC outperforms existing methods in terms of latency reduction, energy efficiency, successful task completion ratio, and cache hit ratio, confirming the benefit of jointly optimizing offloading and resource allocation, with result caching.

Collectively, these contributions provide a robust, cost-efficient, and adaptive framework for UAV-aided MEC, addressing the key challenges of latency, energy consumption, and resource utilization.

The rest of this paper is structured in the following manner. Section 2 summarizes existing studies and identifies the corresponding research gap. Section 3 presents the system model including the network, task, communication, computation, and caching components. Section 4 defines the considered optimization framework. Section 5 introduces the proposed JORC methodology and associated algorithms. Section 6 presents simulation-based performance analysis and compares the proposed JORC framework with baseline approaches. Section 7 discusses the experimental results, outlines the assumptions and limitations of the proposed framework, and identifies promising directions for future research. Lastly, Section 8 presents the concluding remarks and highlights potential future research directions.

2. Related Works

The increasing demand for latency-sensitive and computation-intensive applications has driven extensive research in UAV-aided MEC networks. Prior studies have explored task offloading, resource allocation, and caching from different perspectives, but significant challenges remain in effectively combining these components under UAV-BS collaborative environments. This section reviews the existing studies, highlights key advancements, and identifies the gaps that our proposed JORC framework aims to address.

Computation offloading in MEC: Task offloading is a key component of MEC systems, enabling devices with limited computational power to execute complex tasks by offloading them onto edge or cloud servers. Various studies have explored heuristic, game-theoretic, and deep-learning-based approaches for optimizing offloading decisions. In [9], the authors proposed the JDACO framework in post-disaster UAV-assisted IoT environments, leveraging a VD3QN-based multi-agent RL model to minimize latency and energy consumption. Ref. [16] introduced an AI-driven aerial MEC framework for jointly optimizing offloading, device association, and UAV trajectory planning using a knapsack-based task selection strategy. A layered MEC architecture with edge UAVs and a link UAV was also proposed in paper [17], employing the AGSP algorithm to optimize offloading and UAV deployment. More recently, cooperative multi-UAV MEC studies have jointly optimized task offloading, computation-resource allocation, and trajectory design to reduce latency and energy consumption [18], while service-capacity-oriented UAV-MEC frameworks have emphasized deadline-constrained task completion under limited UAV resources [19]. While these works advance offloading strategies, they generally do not exploit task-result caching for avoiding repeated computation in UAV-BS collaborative MEC systems.

Offloading and resource allocation in MEC: Efficient management of resources such as bandwidth and CPU frequency is critical for ensuring low latency and high throughput in UAV-aided MEC environments. Several studies have integrated offloading decisions with resource allocation, but lack caching mechanisms, which can significantly reduce computational redundancy. In [20], the authors proposed a cloud-edge UAV framework where a hybrid SS-DQN algorithm coordinates task offloading with resource allocation. Ref. [3] introduced a hybrid DRL-greedy algorithm for vehicular networks to allocate bandwidth and CPU resources based on task demands and mobility. In hierarchical UAV systems, a MAPPO-based reinforcement learning model is applied in [21] for optimizing offloading ratios and multi-dimensional resource allocation. In [22], the authors combined GRU-attention mechanisms in a multi-agent setup for optimizing offloading while allocating computing and communication resources. Recent DRL-based UAV-MEC studies

have also used PPO or DQN-style learning to jointly optimize offloading, trajectory, bandwidth, and computation-resource allocation in dynamic aerial edge environments [23,24]. Most recently, the authors in [25] proposed a multi-agent TRPO-based hierarchical offloading framework for large-scale UAV networks. However, these approaches mainly focus on offloading and resource-control decisions and do not integrate the completed task-result reuse through adaptive cache replacement.

Offloading and caching in MEC: Caching can significantly reduce redundant computations and improve system responsiveness by storing frequently accessed data. Several studies have integrated offloading and caching, but they overlook resource allocation optimization. A federated DRL-based model is proposed in [26] for UAV-assisted vehicular networks, enabling dynamic caching and offloading with local model training. In [13], the authors applied DDGP for offloading decisions together with cooperative caching scheme in multi-region MEC environments to enhance efficiency. A hierarchical DRL model is also developed in [27] to jointly manage caching and offloading in edge-cloud systems, utilizing DQN for service caching and SAC for offloading decisions. Cooperative caching and offloading strategies have also been proposed in [14] for mobile augmented reality and vehicular networks, optimizing cache placement and offloading via dynamic programming and artificial bee colony algorithms. Recent UAV-assisted vehicular networks and Internet of vehicles (IoV) studies have further combined task offloading with fuzzy caching, service caching, or content caching using hierarchical DRL, A3C, or DDGP-based methods [28–30]. The authors in [31] combined multi-agent DRL with federated-learning-based caching for UAV-assisted IoV. In addition, multi-UAV and UAV-BS offshore MEC systems have considered service-instance caching, user-preference-oriented service caching, and service retrieval from BS/cloud nodes to reduce service delay and backhaul dependence [32–34]. Nevertheless, these studies mainly cache contents, service programs, or service instances rather than the completed computation results.

Offloading, resource allocation, and caching: A few studies have attempted joint optimization of offloading, resource assignment, and caching but failed to consider UAV-BS collaboration, limiting their adaptability to UAV-aided MEC networks. In [15], the authors proposed a collaborative framework where mobile users offload tasks to MEC servers while cached results are reused to reduce delay and energy. It jointly determines offloading, caching, and allocation of CPU frequency and transmission power. A meta-reinforcement learning approach had also been introduced in [35] to simultaneously optimize offloading, caching, and resource allocation under dynamic user preferences. The approach explored a two-tier network architecture, comprising edge nodes and mobile devices. It employed an RNN-based sequence-to-sequence (seq2seq) network to model the policy and value functions, transforming the problem of solving multiple MDPs into a task execution sequence prediction. A three-tier MEC architecture is also developed in [36], applying a DDQN-based model to coordinate offloading, service caching, and resource allocation across user devices, edge servers, and the cloud layer. In [37], the authors jointly optimized offloading, cache resource assignment, and routing in a satellite-UAV space-air-ground network using program caching, and the authors in [38] jointly optimized offloading, resource allocation, and service caching for multi-UAV MEC using a dual-time-scale TD3 algorithm. Furthermore, service experience-oriented optimization frameworks were proposed in [39] to adjust caching placement, UAV trajectories, and resource distribution to maximize fairness and minimize latency. Both [36,39] focused on caching, but they investigated service caching rather than result caching. Recent UAV-assisted MEC studies have also optimized content caching, computation offloading, and channel allocation under multi-task parallel execution [40], and jointly considered service caching, task offloading, CPU allocation, UAV scheduling, and service delivery for latency-critical UAV networks [41]. However, these approaches still focus on content/service/program caching

and do not directly reuse the completed task-results for repeated or similar computation requests.

Beyond learning-based methods, conventional approaches have also been used to address joint resource allocation and caching problems in MEC. Some studies adopt iterative or metaheuristic techniques, such as the non-DRL iterative framework in [15] and the dynamic programming and artificial-bee-colony-based approach in [14]. Others formulate the problem as a mixed-integer nonlinear optimization (MINLP) program, solved via convex relaxation, alternating optimization, or Lagrangian decomposition. For instance, Yang et al. [42] addressed energy minimization for binary computation offloading in an IRS-aided MEC system using a penalty-based conventional optimization approach. Similarly, Chen et al. [43] jointly optimized hybrid beamforming and content placement in an RIS-aided edge-caching system using an alternating optimization technique. These approaches provide strong optimality guarantees for a given network realization, but generally require the optimization problem to be resolved whenever channel conditions, task arrivals, or cache states change, which limits their scalability in continuously evolving UAV-aided MEC environments with time-varying task arrivals and cache occupancy.

While previous studies have made significant contributions in task offloading, resource allocation, and caching, they fail to integrate all three aspects dynamically under UAV-BS collaboration. Most studies have optimized only two components, resulting in inefficient resource utilization. Studies without UAV-BS collaboration fail to efficiently manage task distribution and resource sharing. Existing DRL-based solutions use DQN, DDQN, A3C, DDPG, or PPO-style models, which may face limitations in exploration stability, convergence behavior, or scalability when handling continuous offloading and resource allocation decisions in dynamic UAV environments. Most caching techniques focus on content or service caching, whereas our framework optimizes task-result caching to reduce redundant computations.

To address these gaps, we propose the JORC framework, that jointly coordinates task offloading and resource allocation decisions using a SAC-based DRL approach and integrates a hybrid LFU-LRU caching strategy to reduce computational redundancy. Table 1 summarizes representative studies on task offloading, resource allocation, and caching in UAV-aided MEC networks, not only comparing their coverage of each component, caching type, UAV-BS collaboration, and solution method but also highlighting how each differs from the proposed JORC framework. Unlike [15], which uses a non-DRL optimization method, and [35], which employs a meta-reinforcement learning approach with an RNN-based seq2seq model, JORC leverages SAC's entropy regularization and off-policy learning, which offer faster convergence and better exploration–exploitation balance in continuous action spaces. Unlike [36,39], which focus on service caching (i.e., caching pre-deployed application services), JORC targets task-result caching, where the computed outputs of specific task instances are stored and reused to directly avoid redundant computation and transmission. Furthermore, none of the existing approaches [15,35,36,39] operate within a UAV-BS collaborative architecture where UAVs with limited capacity dynamically offload tasks to a more powerful BS, which is a key architectural distinction of JORC. By combining SAC-driven offloading and resource allocation control, UAV-BS cooperation, and adaptive result caching, JORC provides a scalable and effective solution for UAV-aided MEC under dynamic network conditions.

Table 1. Comparison of the proposed JORC framework with representative related studies.

Ref.	Year	Offloading	Resource Allocation	Caching Type	UAV-BS Collaboration	Solution Method	Contribution/Limitation
[15]	2021	✓	✓	Task result	—	Conventional	No learning-based adaptability; no UAV involvement

[42]	2022	✓	✓	—	—	Conventional	Requires resolving the optimization problem for each network instance
[35]	2022	✓	✓	Task result	—	Meta-RL	Two-tier MEC only; no continuous entropy-regularized control
[36]	2023	✓	✓	Service	—	DDQN	Discrete-action DQN-based control; service caching; no result reuse
[26]	2024	✓	✓	Content	UAV-assisted (IoV)	Federated DRL	No transmit power allocation; reusable task-result caching
[29]	2025	✓	✓	Content	UAVs, no BS	DDPG	Content caching; no BS collaboration
[18]	2026	✓	✓	—	Cooperative multi-UAV	Conventional	No caching component; no collaboration with BS
[40]	2026	✓	✓	Content	Multi-UAVs	DQN	Content caching rather than reusable task-result caching
Proposed JORC	2026	✓	✓	Task result	✓	SAC (entropy-regularized, continuous control) + hybrid LFU-LRU	Joint optimization of task offloading, resource allocation, and adaptive task-result caching within a UAV-BS collaborative architecture

Note: “✓” means “considered”, and “—” means “not considered”.

3. System Model

This section provides a thorough description of the system model, covering the network, communication, task, offloading, computation, and caching components. Table 2 summarizes the main notations used throughout the paper.

Table 2. List of notations used in this study.

Notation	Definition	Notation	Definition
N	Set of UDs	$L_{n,m,b}^{off}$	Uplink transmission latency (UAV to BS)
M	Set of UAVs	$E_{n,m,b}^{off}$	Uplink transmission energy consumption (UAV to BS)
H	UAV altitude	L_n^{UD}	Latency for computing on UD
$h_{n,m}$	Channel gain	L_n^{UAV}	Latency for computing at UAV
$d_{n,m}$	Horizontal distance between UD n to UAV m	L_n^{BS}	Latency for computing at BS
B	Bandwidth	E_n^{UD}	Energy consumption for computing on UD
N_0	Noise power	E_n^{UAV}	Energy consumption for computing at UAV
P_n	Transmit power of UD n	E_n^{BS}	Energy consumption for computing at BS
P_m	Transmit power of UAV m	f_n	Computing ability of UD (CPU cycles/s)
T_n	Task information of UD n	F_u	Computing ability of UAV (CPU cycles/s)
L_n^{max}	Maximum tolerable latency	F_b	Computing ability of BS (CPU cycles/s)
D_n^{in}, W_n, D_n^{out}	Task size, Computing workload, Result size	k	Energy conversion efficiency
$L_{n,m}^{off}$	Uplink transmission latency (UD to UAV)	S_m^{max}	Maximum caching capacity of MEC m
$E_{n,m}^{off}$	Uplink transmission energy consumption (UD to UAV)	S_b^{max}	Maximum caching capacity of BS

3.1. Network Model

The system under consideration consists of a single BS and multiple UAVs, where both the BS and UAVs are equipped with edge servers to support task execution. Formally, the BS is represented as b and the set of UAVs is represented as $m \in M = \{1, 2, 3, \dots, M\}$. Additionally, the system includes N user devices (UDs), such as smartphones, AR, VR, and IoT devices, each requiring computational resources for task execution. The set of UD is represented as $n \in N = \{1, 2, 3, \dots, N\}$. The spatial distribution of BS, UAVs and UD is represented using a 3D Cartesian coordinate system. Since the BS is located on the ground, its position is defined as $q_b = (0, 0)$. For UAV m , the horizontal position is defined by $q_m = (x_m, y_m)$, where $m \in M$. Each UAV is equipped with a directional antenna with beamwidth θ and operates at a fixed altitude H . The height of user devices is negligible in comparison to the altitude of the UAVs. The horizontal coordinate of user device n is defined by $q_n = (x_n, y_n)$, where $n \in N$. A quasi-static scenario is considered where UAVs and user devices remain stationary within a task offloading period, although their positions may vary across different periods [44].

3.2. Communication Model

UAVs establish wireless backhaul connections with a BS, and separate frequency bands are allocated for fronthaul and backhaul communications to reduce signal interference [44]. The fronthaul links connect UD to the UAVs, while the backhaul links connect the UAVs to the BS. For ease of modeling, the backhaul capacity from the BS to each UAV was considered as $R_{m,b}$. To mitigate co-channel interference over fronthaul transmissions, OFDMA (i.e., orthogonal frequency-division multiple access) is adopted [15]. Considering the elevation of UAVs, we consider the wireless connections between UAVs and devices to be LoS. Because of the mobility consideration as a quasi-static scenario, we may ignore the Doppler Effect. Therefore, the channel gain between device n and UAV m is modeled as

$$h_{n,m} = \frac{h_0}{H^2 + d_{n,m}^2}, \quad (1)$$

where h_0 is the reference channel gain at 1 m, and $d_{n,m} = \|q_m - q_n\| = \sqrt{(x_m - x_n)^2 + (y_m - y_n)^2}$, which is the horizontal distance between the UAV m and UD n .

The service radius of UAV m is defined by $H \tan \theta$, where H denotes the UAV's altitude and θ denotes the antenna beamwidth angle. Therefore, for a UD n to offload its task to UAV m , the horizontal distance $d_{n,m}$ must not exceed $H \tan \theta$ (i.e., $d_{n,m} \leq H \tan \theta$). Accordingly, the uplink transmission rate of device n to UAV m is given by

$$R_{n,m} = B_n \log_2 \left(1 + \frac{P_n h_{n,m}}{B_n N_0} \right), \quad (2)$$

where P_n represents the power transmitted by UD n , N_0 denotes the spectral density of noise at the receiver, B_n represents the bandwidth assigned to UD n .

3.3. Task Model

Each UD generates computational tasks that must be either executed locally or offloaded to a UAV or the BS. A computational task of UD n at time t is characterized by a tuple (T_n, L_n^{max}) , where T_n denotes the task information, and L_n^{max} denotes the maximum tolerable latency for that task. Task information T_n is represented as $(D_n^{in}, W_n, D_n^{out})$; where D_n^{in} denotes the task size, W_n denotes the CPU cycles required (computing workload) to compute the task, and D_n^{out} represents the size of the result after the completion of the task. To ensure QoS, each task must be executed within its latency deadline.

3.4. Offloading Model

A UD n can send its computational task to the MEC server deployed at UAV m or the BS. The offloading decision variable $X = \{x_n^{UD}, x_{n,m}^{UAV}, x_n^{BS}\} \in \{0, 1\}$, where $n \in N$ and $m \in M$. $x_n^{UD} = 1$ means that the task is executed locally at UD n , $x_{n,m}^{UAV} = 1$ indicates the task is assigned to UAV m , and $x_n^{BS} = 1$ indicates that the task is further offloaded to BS for execution, otherwise 0. To ensure mutual exclusivity in task execution, only one offloading decision is permitted per task, satisfying the following constraint:

$$x_n^{UD} + \sum_{m \in M} x_{n,m}^{UAV} + x_n^{BS} = 1. \quad (3)$$

If the task is offloaded, the system must account for both uplink and downlink transmission latency and energy consumption.

Uplink transmission latency (UD to UAV): The uplink transmission latency occurs when UD n transmits the task input data D_n^{in} to UAV m . For this offloading case, the latency is expressed as

$$L_{n,m}^{off} = \frac{D_n^{in}}{R_{n,m}}, \quad (4)$$

where $R_{n,m}$ is the uplink data rate from UD n to UAV m ; otherwise, the latency is zero.

Uplink transmission energy consumption (UD to UAV): the corresponding uplink energy consumed for this transmission is expressed as

$$E_{n,m}^{off} = \frac{P_{n,m} D_n^{in}}{R_{n,m}}, \quad (5)$$

where $P_{n,m}$ is the transmit power of UD n .

Uplink transmission latency (UAV to BS): if UAV m further forwards the task to the BS for computation, then the uplink transmission latency is expressed as

$$L_{n,m,b}^{off} = \frac{D_n^{in}}{R_{m,b}}, \quad (6)$$

where $R_{m,b}$ represents the backhaul capacity.

Uplink transmission energy consumption (UAV to BS): The energy required for this UAV to BS transmission is computed as

$$E_{n,m,b}^{off} = \frac{P_{m,b} D_n^{in}}{R_{m,b}}, \quad (7)$$

where $P_{m,b}$ is the uplink transmit power of UAV m . If the task is executed locally on UD, no uplink transmission is required; therefore, the corresponding delays and energy terms are zero.

Downlink transmission latency and energy consumption: due to the relatively small size of task-results and the MEC to UD downlink rates being typically high, the downlink transmission latency and energy consumption are considered negligible and, thus, are omitted from the analysis [14,15].

3.5. Computation Model

Computation task T_n of UD n can be either computed locally on UD n , or remotely at UAV m , or the BS.

Computing locally on UDs: When the task T_n is processed by UD n , then the computation latency is given by

$$L_n^{UD} = \frac{W_n}{f_n}, \quad (8)$$

where W_n denotes the computing workload (required CPU cycles) of task T_n and f_n denotes the local computing ability in CPU cycles per second of UD n . The corresponding energy consumption is expressed as

$$E_n^{UD} = k f_n^2 W_n, \quad (9)$$

where k is the energy-conversion coefficient determined by the processor chip design.

Computing at UAVs: When the task T_n is processed at UAV server m , then the latency is calculated as

$$L_n^{UAV} = \frac{W_n}{s_{m,n} F_u}, \quad (10)$$

where W_n represents the computing workload (required CPU cycles) of task T_n , F_u represents the total computing ability in CPU cycles per second of UAV m , and $s_{m,n} \in [0, 1]$ represents the percentage of UAV m 's computation resource assigned to UD n . The corresponding energy consumption is calculated as

$$E_n^{UAV} = k (s_{m,n} F_u)^2 W_n, \quad (11)$$

where k is the energy-conversion coefficient determined by the processor chip design.

Computing at BS: When the task T_n is processed at the BS, then the latency is expressed as

$$L_n^{BS} = \frac{W_n}{s_{b,n} F_b}, \quad (12)$$

where W_n denotes the computing workload (required CPU cycles) of task T_n , F_b denotes the total computing ability (CPU cycles per second) of the BS, and $s_{b,n} \in [0, 1]$ represents the percentage of the BS's computation resource assigned to UD n . The corresponding energy consumption is calculated as

$$E_n^{BS} = k (s_{b,n} F_b)^2 W_n, \quad (13)$$

where k is the energy-conversion coefficient determined by the processor chip design.

3.6. Caching Model

In the proposed system, caching refers to the process through which a MEC server (either a UAV or the BS) stores the output of a completed task together with its associated data. When a user device (UD) later requests to execute the same task with an identical input, the MEC server compares the new request with the stored entries and identifies it as a cache hit. In this case, the UD does not need to offload the task again, and the MEC server avoids recomputation. Instead, the previously stored result is returned directly to the UD. This approach reduces transmission delay, computation delay, and overall energy consumption.

The proposed framework automatically caches the result of each newly executed offloaded task. Since MEC servers have limited storage capacity, not all task-results can be stored simultaneously. To satisfy the storage constraint, the total amount of data cached at MEC server i must remain within its maximum storage limit S_i^{max} . That is,

$$\sum_{n \in N} x_n^i \cdot D_n^{out} \leq S_i^{max}, \quad \forall i \in M \cup \{b\}, \quad (14)$$

where S_i^{max} denotes the available storage limit at MEC server i . This equation ensures that the total cached data at MEC server i does not exceed its storage limit.

In our proposed framework, the cache lookup is performed based on a unique task identifier. When the same computational task with identical input is requested again, the system recognizes it as a cache hit and retrieves the stored result directly, bypassing

recomputation [15]. This assumption is well-suited for applications that generate deterministic and repetitive computational requests, such as fixed AR scene rendering, repeated sensor data inference in IoT networks, or periodic image-classification tasks in autonomous systems [14,15].

4. Problem Formulation

This study aims to minimize the overall cost of a UAV-aided MEC system. Cost is modeled as a weighted combination of latency and energy consumption. Because the units of latency and energy are different, we used the normalized latency and energy values.

In the proposed system, a cache hit avoids task offloading and computation, allowing the result to be delivered directly to the user. Only cache query latency and energy are involved, which are negligible compared to offloading and computation costs. In contrast, a cache miss requires full computation and communication latency and energy to be considered.

The total latency associated with a single task is calculated as follows:

$$L_n = x_n^{UD}(L_n^{UD}) + x_{n,m}^{UAV}(L_{n,m}^{off} + L_n^{UAV}) + x_n^{BS}(L_{n,m}^{off} + L_{n,m,b}^{off} + L_n^{BS}). \quad (15)$$

Similarly, the total energy required for a single task is expressed as follows:

$$E_n = x_n^{UD}(E_n^{UD}) + x_{n,m}^{UAV}(E_{n,m}^{off} + E_n^{UAV}) + x_n^{BS}(E_{n,m}^{off} + E_{n,m,b}^{off} + E_n^{BS}). \quad (16)$$

Hence, the cost for a single user can be defined as

$$C_n = \gamma_n^L \cdot L'_n + \gamma_n^E \cdot E'_n, \quad (17)$$

where L'_n and E'_n denote the normalized values of latency and energy consumption, respectively, and $L'_n = L_n/L_n^{max}$ and $E'_n = E_n/E_n^{max}$. L_n^{max} and E_n^{max} are the maximum values of L_n and E_n respectively. γ_n^L and γ_n^E are the weight factors of latency and energy consumption, respectively. The range of these weighted factors are $0 \leq \gamma_n^L \leq 1$ and $0 \leq \gamma_n^E \leq 1$, respectively, where $\gamma_n^L + \gamma_n^E = 1$.

Therefore, the overall cost for all users can be defined as

$$C = \sum_{n=1}^N C_n. \quad (18)$$

Our goal is to minimize the overall cost of the system. Hence, the optimization problem can be expressed as follows:

$$\min_{X,P,F} C(X,P,F) \quad (19a)$$

subject to:

$$X = \{x_n^{UD}, x_{n,m}^{UAV}, x_n^{BS}\} \in \{0, 1\}, \forall n \in N, \forall m \in M \quad (19b)$$

$$x_n^{UD} + \sum_{m \in M} x_{n,m}^{UAV} + x_n^{BS} = 1, \quad \forall n \in N, \forall m \in M \quad (19c)$$

$$L_n \leq L_n^{max} \quad (19d)$$

$$0 < P \leq P^{max} \quad (19e)$$

$$\sum_{n=1}^N s_{i,n} F_i \leq F_i, \forall i \in M \cup \{b\}. \quad (19f)$$

where X represents the set of offloading decision variables, P denotes the transmit power allocation variable, and F denotes the variable of computing frequency allocated by the MEC servers (UAVs and the BS). Constraint (19b) defines the binary offloading decisions, whereas (19c) ensures that each task is computed at exactly one location (UD, UAV, or BS). Constraint (19d) requires the completion latency of each task to remain within its maximum tolerable threshold. Constraint (19e) guarantees that the allocated transmit power is strictly positive and does not exceed the maximum allowable limit. Finally, constraint (19f) restricts the total CPU frequency allocation of all tasks within the computing capacity of each MEC server.

5. Joint Task Offloading and Resource Allocation with Data Caching (JORC) Scheme

This section explains the design and operational flow of the proposed JORC algorithm, which aims to reduce the overall cost of a UAV-BS collaborative system. JORC integrates a SAC-based DRL mechanism for offloading decisions and resource allocation, alongside a hybrid LFU-LRU caching strategy to improve storage efficiency. The architecture of the proposed JORC scheme is illustrated in Figure 2.

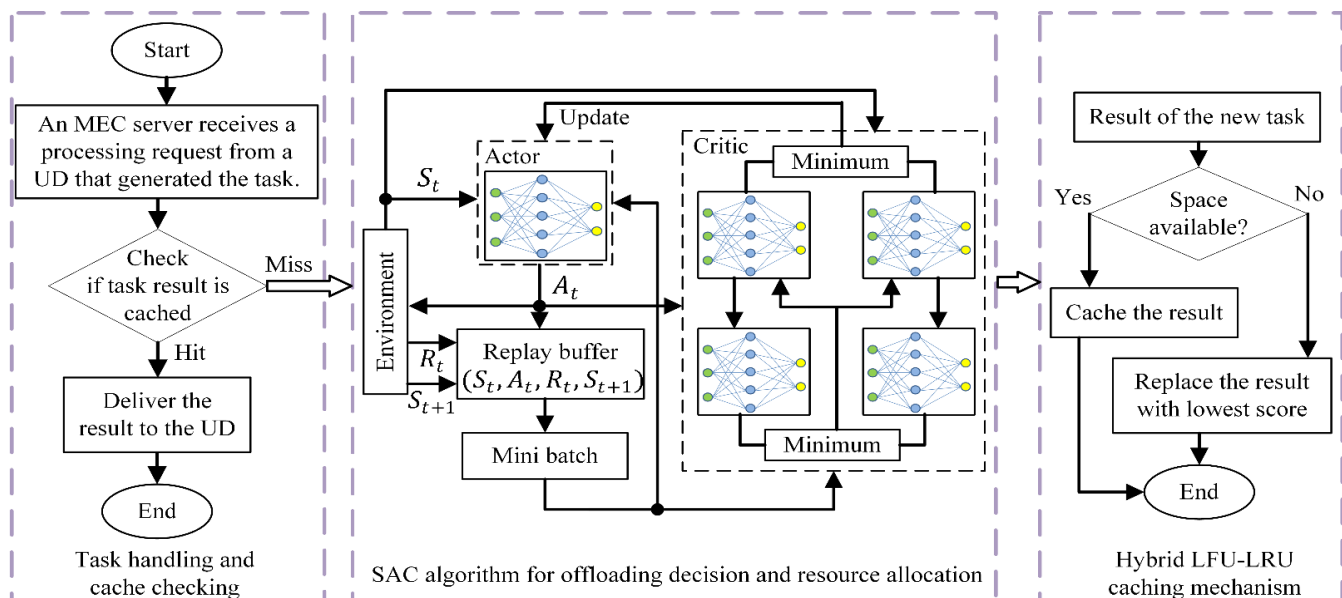


Figure 2. Architecture of our proposed JORC scheme consisting of (i) task handling and cache checking, (ii) SAC algorithm for offloading decision and resource allocation, and (iii) hybrid LFU-LRU caching mechanism (All the operations in Figure 2 are carried out by the MEC servers (i.e., UAVs and BS)).

The JORC framework operates sequentially to optimize real-time decision making in UAV-BS collaborative networks. When a UD generates a task, an offloading request is first sent to the UAV or BS. The UAV or BS then checks whether the requested task-results have been cached. If a cache hit occurs, the result is retrieved and delivered immediately to the UD, thereby eliminating repeated computation. For a cache miss, the SAC-based decision making process determines the optimal execution location and allocates computational and communication resources. Once the task has been executed, the result is stored in the cache using the LFU-LRU policy, making it available for immediate reuse in future requests.

5.1. SAC Algorithm for Joint Offloading and Resource Management

Although the communication model adopts a quasi-static large-scale channel model, the overall UAV-aided MEC system remains dynamic because user locations, task arrivals, available computing resources, and workload distribution vary over time. Consequently, the JORC framework must repeatedly determine task offloading and resource allocation decisions as these conditions evolve. The underlying offloading and resource allocation problem is formulated as a non-convex mixed-integer nonlinear optimization problem. Solving it online repeatedly as the system state changes can be computationally demanding, particularly under the strict per-task latency constraints considered in this study. Therefore, the proposed SAC-based DRL framework learns an adaptive decision policy offline, which can subsequently be applied efficiently online without repeatedly solving the optimization problem.

The choice of SAC over other DRL algorithms is motivated by the specific characteristics of the joint offloading–resource management task in UAV-aided MEC. First, the action space of JORC is continuous and high-dimensional, comprising offloading decisions, transmission power allocation, and CPU frequency allocation simultaneously. SAC is inherently designed for continuous action spaces through its stochastic actor formulation, whereas algorithms such as DQN and DDQN are restricted to discrete action spaces and would require action discretization that introduces approximation errors. Second, relative to on-policy methods such as PPO, SAC’s off-policy learning with a replay buffer significantly improves sample efficiency, which is critical in dynamic MEC environments where poor decisions carry real costs. Third, compared to TD3, which also handles continuous actions, SAC’s entropy regularization explicitly encourages exploration by maintaining policy stochasticity, preventing premature convergence to suboptimal offloading strategies in early training stages. This is particularly valuable in UAV-aided MEC, where the joint state space (user positions, channel conditions, resource availability, and cache state) is large and constantly evolving. The empirical convergence results in Section 6.4.1 further validate this choice, showing that SAC achieves stable convergence approximately 330 episodes faster than PPO and 680 episodes faster than A3C.

5.1.1. MDP Representation

The joint offloading and resource-management problem in the UAV-aided MEC system is represented as an MDP to support adaptive decision making under dynamic network conditions. An MDP is specified by a tuple $(S, A, \mathcal{P}, R, \gamma)$, where S is the state space, A is the action space, $\mathcal{P}$ is the state-transition probability, R is the reward function, and γ is the discount factor that balances immediate and future rewards.

State space (S_t): The state captures the current system condition, including:

$$S_t = \{S_t^{UD}, S_t^{UAV}, S_t^{BS}, S_t^{network}\}, \quad (20)$$

where S_t^{UD} represents the user information (including task information T_n , maximum tolerable latency L_n^{max} , energy status E_n^{UD} and position of the UD (x_n, y_n)), S_t^{UAV} denotes the UAV state (available CPU resources F_m , current workload, and position of the UAV (x_m, y_m)), S_t^{BS} denotes the state of BS (available computing capacity F_b), and $S_t^{network}$ represents the network state (channel gain $h_{n,m}$ between user n and UAV m , and network congestion status).

Action space (A_t): This space defines the control variables used for offloading decisions and resource allocation. At each time step t , action A_t is represented as

$$A_t = \{X, P, F\}. \quad (21)$$

Because the offloading decision X in (3), (19b), and (19c) is binary and mutually exclusive, whereas the SAC actor produces continuous outputs, a projection step is used to

obtain a feasible offloading decision. For each user n , the actor outputs continuous-valued preference scores for each candidate execution location, $z_n = [z_n^{UD}, z_n^{UAV}, z_n^{BS}]$. The execution location is selected as

$$i_n^* = \arg \max_{i \in \{UD, UAV, BS\}} z_n^i, \quad (21a)$$

and the corresponding binary offloading decision is obtained as

$$x_n^i = \begin{cases} 1, & i = i_n^*, \\ 0, & \text{otherwise}, \end{cases} \quad (21b)$$

thereby satisfying the binary and mutual exclusivity constraints in (19c). The remaining action components, namely transmit power and CPU frequency allocation, remain continuous.

Reward (R_t): SAC aims to maximize the accumulated reward, whereas our target is to minimize cost. Therefore, the reward is set as the negative system cost, as follows:

$$R_t = -C(t). \quad (22)$$

Transition probability ($\mathcal{P}_t$): The state transition probability $\mathcal{P}_t(S_{t+1}|S_t, A_t)$ describes the probability of reaching state S_{t+1} from the current state S_t , when action A_t is chosen. In this work, the transition probability is not assumed to be known in closed form and is neither explicitly modeled nor estimated. Instead, the proposed SAC algorithm learns directly from the sampled transitions (S_t, A_t, R_t, S_{t+1}) generated through interaction with the environment, without requiring prior knowledge of the transition dynamics.

5.1.2. Soft Actor–Critic (SAC) Algorithm

In the SAC algorithm, the expected reward and entropy are included in the objective function to maximize. The expected entropy ensures stability and exploration through random actions. The main goal is to learn an optimal policy π^* that maximizes both the expected reward and entropy, defined as

$$\pi^* = \arg \max_{\pi} \sum_t \mathbb{E}[R_t(S_t, A_t) + \alpha H(\pi(\cdot | S_t))], \quad (23)$$

where $\mathbb{E}[\cdot]$ denotes the expectation of a function, $H(\pi(\cdot | S_t))$ represents entropy of policy π in state S_t , and α represents the temperature parameter that defines the relative significance of entropy and controls the stochasticity of the policy. The temperature parameter should be dynamic because the policy gradually improves with experience; therefore, less exploration is needed when there is a clear distinction between poor and optimal actions. The dynamic entropy adjustment in SAC was first proposed in [45]. This is crucial, as it allows the policy to adapt dynamically, ensuring sufficient exploration when the best action is ambiguous, while becoming less stochastic in states where the preferred action is evident. This dynamic adjustment improved the performance and stability of the learning algorithm. We considered this dynamic temperature adjustment when designing the SAC algorithm.

Algorithm 1 describes the training procedure of the proposed SAC model. The SAC algorithm has two critic networks, Q_1 and Q_2 with parameters θ_{Q_1} and θ_{Q_2} , two target critic networks, Q'_1 and Q'_2 with parameters θ'_{Q_1} and θ'_{Q_2} , and actor-network π with parameter θ_{π} . The agent observes the environmental state S_t , then selects an action A_t according to the current policy π . After executing the action, the agent receives a reward $R_t(S_t, A_t)$ and transitions to the next state S_{t+1} , and then stores the transition tuple (S_t, A_t, R_t, S_{t+1}) in the replay buffer $\mathcal{D}$.

Algorithm 1. SAC algorithm for task offloading and resource-management decisions.

Input: Maximum number of episodes, maximum number of steps in each episode, environment state information S_t , reward function $R_t(S_t, A_t)$, learning rates λ , discount factor γ , initial temperature parameter α , and constant ε .

Output: Optimal X, P, and F.

```

1: Initialize actor network  $\pi(A|S; \theta_\pi)$  with parameters  $\theta_\pi$ 
2: Initialize two critic networks  $Q_1(S, A; \theta_{Q_1})$  and  $Q_2(S, A; \theta_{Q_2})$  with parameters  $\theta_{Q_1}$ 
   and  $\theta_{Q_2}$ , and two target critic networks  $Q'_1$  and  $Q'_2$  with parameters  $\theta_{Q'_1} \leftarrow \theta_{Q_1}$ ,
    $\theta_{Q'_2} \leftarrow \theta_{Q_2}$ 
3: Initialize temperature parameter  $\alpha$ , replay buffer  $\mathcal{D}$ , target entropy  $H$ 
4: for each episode do
5:   Reinitialize environment and receive initial state  $S_0$ 
6:   for each time step in an episode do
7:     Choose action  $A_t$  based on current policy and execute it
8:     Obtain a reward  $R_t(S_t, A_t)$  and a next state  $S_{t+1} \sim P_{tn}(S_{t+1}|S_t, A_t)$ 
9:     Append transition tuple  $(S_t, A_t, R_t, S_{t+1})$  to the replay buffer  $\mathcal{D}$ 
10:    Draw a mini batch of samples  $(S_t, A_t, R_t, S_{t+1})$  from  $\mathcal{D}$ 
11:    update critic parameters based on (27)
        $\theta_{Q_i} \leftarrow \theta_{Q_i} - \lambda_Q \nabla_{\theta_{Q_i}} J_Q(\theta_{Q_i})$  for  $i \in \{1, 2\}$ 
12:    update actor parameters based on (30)
        $\theta_\pi \leftarrow \theta_\pi - \lambda_\pi \nabla_{\theta_\pi} J_\pi(\theta_\pi)$ 
13:    update temperature parameter by minimizing (31)
        $\alpha \leftarrow \alpha - \lambda_\alpha \nabla_\alpha J(\alpha)$ 
14:    update target critic network parameters by (32)
        $\theta_{Q'_i} \leftarrow \varpi \theta_{Q_i} + (1 - \varpi) \theta_{Q'_i}$  for  $i \in \{1, 2\}$ 
15:   end for
16: end for

```

Policy evaluation, policy improvement, and automatic temperature parameter adjustment occurred during the SAC training phase.

To evaluate the soft policy, first we need to calculate soft Q values, which are computed as

$$Q(S_t, A_t) = R_t(S_t, A_t) + \gamma \mathbb{E}_{S_{t+1} \sim P_{tn}}[V(S_{t+1})], \quad (24)$$

where γ denotes the discount factor and $V(S_{t+1})$ represent the soft value function for state S_{t+1} , which can be defined as

$$V(S_t) = \mathbb{E}_{A_t \sim \pi}[Q(S_t, A_t) - \alpha \log \pi(A_t|S_t)]. \quad (25)$$

The parameter θ of policy evaluation network (critic) Q is updated to minimize the loss function represented as

$$J_Q(\theta) = \mathbb{E}_{(S_t, A_t) \sim \mathcal{D}} \left[\frac{1}{2} \left(Q(S_t, A_t) - \left(R_t(S_t, A_t) + \gamma \mathbb{E}_{S_{t+1} \sim P_{tn}} [V_{\theta_Q}(S_{t+1})] \right) \right)^2 \right], \quad (26)$$

where the soft state value function is calculated using Equation (24) with target parameters $\theta_{Q'}$. We use gradient descent with mini-batch samples to update θ :

$$\nabla_\theta J_Q(\theta) = \nabla_\theta Q_\theta(S_t, A_t) \left(Q_\theta(S_t, A_t) - \left(R_t(S_t, A_t) + \gamma \left(Q_{\theta_{Q'}}(S_{t+1}, A_{t+1}) - \log(\pi(A_{t+1}|S_{t+1})) \right) \right) \right), \quad (27)$$

Next, in the policy improvement stage, parameters θ_π of policy network (actor) π is updated by minimizing the expected KL-divergence:

$$J_{\pi}(\theta_{\pi}) = \mathbb{E}_{(S_t, A_t) \sim D} \left[\mathbb{E}_{A_t \sim \pi} [\alpha \log \pi(A_t | S_t) - Q(S_t, A_t)] \right]. \quad (28)$$

The details of obtaining (28) using the KL divergence are provided in [46]. To calculate the gradient descent of (27), we need to reparametrize A_t , which helps to reduce the gradient estimation variance during policy optimization. We begin by selecting a random sample ψ_t from a predetermined distribution (e.g., standard normal distribution). Next, we derive A_t as

$$A_t = \mu_t + \psi_t \odot \varepsilon_t, \quad (29)$$

where μ_t represents the mean of the policy, ε_t denotes the standard deviation of policy π , and $\odot$ signifies the Hadamard product. Then, the gradient update is calculated as

$$\nabla_{\theta_{\pi}} J_{\pi}(\theta_{\pi}) = \nabla_{\theta_{\pi}} \alpha \log \pi(A_t | S_t) + \left(\nabla_{A_t} \alpha \log \pi(A_t | S_t) - \nabla_{A_t} Q(S_t, A_t) \right) \nabla_{\theta_{\pi}} A_t. \quad (30)$$

We use two critic networks, each parameterized by $\theta_{Q_i}, i \in \{1, 2\}$, to mitigate positive bias during policy improvement, which can degrade algorithm performance. By independently training the networks to optimize $J_Q(\theta_{Q_i})$ and using the minimum Q-value in (27) and (30), the clipped double-Q learning approach reduces overestimation bias, enhances the stability and accuracy of value function estimates, and improves overall learning [47,48].

Next, we need to adjust the temperature parameter α . A larger value of α increases stochasticity for exploration, while a gradually decreasing temperature encourages more deterministic actions as the policy converges, focusing on exploiting the learned optimal actions. Thus, we adjust α by minimizing the following function:

$$j(\alpha) = \mathbb{E}_{A_t \sim \pi} [-\alpha \log \pi(A_t | S_t) - \alpha \bar{H}], \quad (31)$$

where $\bar{H}$ represents the entropy threshold.

Finally, the target network weights were updated by considering the moving average of the critic network weights. Specifically, the target network parameters $\theta_{Q'_i}$ are updated as

$$\theta_{Q'_i} \leftarrow \varpi \theta_{Q_i} + (1 - \varpi) \theta_{Q'_i}, \quad (32)$$

where ϖ denotes a parameter that controls the update rate. This method of updating the target network weights helps stabilize the training process by smoothing out changes in the network parameters.

5.2. Caching Management Subproblem

Efficient caching management is essential for reducing task redundancy and improving resource utilization in UAV-aided MEC. This study adopts a hybrid LFU-LRU caching strategy to dynamically store and manage task-results. The proposed method ensures that frequently requested tasks remain in the cache, while preserving recently accessed tasks, thereby making the system adaptable to different request patterns.

Caching management strategy: When the cache capacity constraint is reached, the MEC server cannot store all computed results. Therefore, a cache replacement decision is required. In the proposed framework, the cached result to be erased is determined by the hybrid LFU-LRU cache score. Each cached task-result entry is assigned a hybrid cache score based on two factors: (1) Access frequency ($\mathcal{F}(T_n)$), which represents the cumulative number of cache hits recorded for the cached task-result entry since it was inserted into the cache. Specifically, each cached entry maintains an access counter, which is incremented whenever an incoming request matches the cached deterministic task-result (i.e., the same task with the same input) and the cached result is successfully reused. Conse-

quently, computation results with higher access frequencies are regarded as more popular. This corresponds to the LFU component. (2) Last access time ($\mathcal{L}(T_n)$), which records the most recent cache hit time of the cached task-result entry and therefore reflects the recency of usage. This corresponds to the LRU component. The cache score is computed as follows:

$$\mathcal{H}(T_n) = \delta \mathcal{F}(T_n) + (1 - \delta) \frac{1}{t - \mathcal{L}(T_n)}, \quad (33)$$

where $\delta \in [0, 1]$ denotes the weighting parameter balancing LFU and LRU. A higher δ favors frequent tasks (LFU). A lower δ prioritizes recently used tasks (LRU). Moreover, $\mathcal{F}(T_n)$ is updated upon every cache hit, whereas $\mathcal{L}(T_n)$ is updated to the current time whenever the cached task-result is accessed. When the cache reaches its storage capacity, the cached task-result entry with the lowest hybrid score is evicted first because it has the lowest combined priority in terms of access frequency and recency. The newly computed task-result is then inserted into the cache if it was executed at a UAV or the BS.

The weighting factor δ is adapted dynamically based on the observed task repetition rate within a sliding time window of size $\mathcal{W}$. The adaptation rule is defined as:

$$\delta(t + 1) = \text{clip}(\delta(t) + \eta(r_t - r_{th}), \delta_{min}, \delta_{max}), \quad (34)$$

where $\eta > 0$ is the adaptation step size, r_t denotes the task repetition rate at time step t , r_{th} is a predefined repetition rate threshold, and δ_{min} and $\delta_{max} \in [0, 1]$ are the lower and upper bounds of δ , respectively. When $r_t > r_{th}$, indicating that task requests are highly repetitive, δ increases, placing more weight on access frequency (LFU) to retain popular tasks in the cache. Conversely, when $r_t < r_{th}$, indicating diverse and non-repetitive requests, δ decreases, placing more weight on recency (LRU) to retain the most recently used tasks. This adaptive mechanism ensures that the caching policy self-tunes to the observed request distribution without requiring manual tuning or prior knowledge of the workload.

5.3. Joint Offloading and Resource Allocation with Caching (JORC) Scheme

The JORC scheme optimizes computational task execution in UAV-aided MEC networks. It uses an SAC-driven DRL method for offloading and resource allocation while managing caching with a hybrid LFU-LRU mechanism, as depicted in Algorithm 2.

Working process of JORC algorithm: The algorithm operates in three main steps:

- i. Task handling and cache checking: When a user requests a task, the system first checks whether the result already exists in the cache. If a task-result is found in the cache (cache hit), it is directly returned to the user. If the result is unavailable in the cache (a cache miss), the system proceeds with offloading and resource allocation.
- ii. SAC-based offloading and resource allocation: The SAC algorithm is used to determine whether a task should be processed locally, at a UAV or BS. SAC dynamically allocates computing and communication resources to balance the network load and minimize delays. Once the task is processed, the result is delivered to the requesting user.

Algorithm 2. JORC algorithm integrating SAC-based offloading and resource allocation with hybrid LFU-LRU caching mechanism.

Input: Set of tasks T_n , network state (channel conditions, congestion, and available resources), cache state (stored tasks, access frequencies, last access times, and capacity), trained SAC model for offloading and resource allocation, and initial repetition rate threshold r_{th} for cache adaptation.

Output: Updated cache storage, updated frequency and last access time for cached tasks, adapted δ for dynamic cache management.

Initialization

```

1: Define cache storage  $S_m$  with a maximum capacity  $S_m^{max}$ .
2: Initialize frequency counter  $\mathcal{F}(T_n) = 0$  for all tasks.
3: Initialize the last access time  $\mathcal{L}(T_n) = 0$  for all tasks.
4: Initialize the repetition rate threshold  $r_{th}$ .
   # Task request handling
5: For each requested task  $T_n$ :
6:     Check the result of  $T_n$  in  $S_m$ 
7:     If Cache Hit (Task Found in Cache):
8:         Serve the task-result from cache storage  $S_m$ .
9:         Update access frequency:  $\mathcal{F}(T_n) \leftarrow \mathcal{F}(T_n) + 1$ .
10:        Update last access time:  $\mathcal{L}(T_n) \leftarrow t$ .
11:    If Cache Miss (Task Not in Cache):
12:        # Need to take offloading decision and resource allocation to process the task
13:        Execute Algorithm 1
14:        Processed the task as per the decision and obtained the result.
15:        If executed at UAV or BS:
16:            If cache storage is full:
17:                Compute hybrid cache scores for all cached tasks using Equation (33).
18:                Identify the task with the lowest score:
19:                     $T_{evict} = \arg \min_{T_j \in S_m} \mathcal{H}(T_n)$ 
20:                Remove  $T_{evict}$  from cache.
21:                Store new task  $T_n$  in cache.
22:                Initialize  $\mathcal{F}(T_n) = 1$  and  $\mathcal{L}(T_n) = t$ .
   # Dynamic cache adaptation
23: Update  $\delta$  by Equation (34):  $\delta(t+1) = clip(\delta(t) + \eta(r_t - r_{th}), \delta_{min}, \delta_{max})$ 

```

iii. Hybrid LFU-LRU caching mechanism: If the caching space is available, the newly processed task-result is stored. If the cache is full, the algorithm removes the least important task by using a hybrid LFU-LRU strategy. The LFU retains frequently requested tasks, and the LRU ensures that recently used tasks are retained. A weighting factor (δ) adaptively balances the importance of frequency and recency based on user request patterns. This adaptive caching mechanism ensures efficient storage use while reducing redundant computations.

5.4. Complexity Analysis

The computational complexity of the proposed JORC framework is analyzed by evaluating the complexity of its key components: the SAC-based offloading and resource allocation algorithm (Algorithm 1) and the caching management process (Algorithm 2).

The SAC algorithm operates over n_e episodes, each consisting of n_s time steps. The complexity of the algorithm is analyzed as follows: Network initialization and parameter setup: $O(1)$, selecting and executing actions: $O(1)$, storing transitions in the replay buffer: $O(1)$, sampling a mini-batch of size k from the replay buffer: $O(k)$, and updating critic and actor parameters using gradient descent: $O(1)$. For each time step, the complexity is $O(k)$. Because each episode consists of n_s steps, the complexity for one episode is $O(n_s \cdot k)$. Over n_e episodes, the total complexity is $O(n_e \cdot n_s \cdot k)$. Because k (mini-batch size) is a small constant, the overall complexity simplifies to: $O(n_e \cdot n_s)$. This complexity reflects the training process of the SAC algorithm, where each iteration involves policy evaluation, policy improvement, and entropy tuning, supporting adaptive offloading and resource-management decisions.

The caching mechanism integrates hybrid LFU and LRU strategies. The complexity of its key operations is: Cache hit update: $O(1)$, computing the hybrid cache score: $O(1)$,

eviction process (removing the lowest-score task): $O(\log C)$ (if a priority queue is used, where C is the cache size), and insertion of a new task into the cache: $O(1)$. Because cache operations involve maintaining a priority queue for eviction, the overall complexity of caching management is $O(\log C)$ per caching update. This ensures efficient cache operations with minimal computational overhead.

Therefore, the overall complexity of the JORC scheme combines the SAC-based offloading and resource allocation algorithm with the caching mechanism. Because the caching operations are performed at each time step, the total complexity is: $O(n_e \cdot n_s \cdot \log C)$.

The primary computational overhead of the proposed framework occurs during the offline training stage of the SAC algorithm. Once training is completed, online decision making only requires a forward pass through the trained actor network, enabling efficient real-time task offloading and resource allocation decisions. As the number of users and the number of UAVs increase, the state and action dimensions also increase, resulting in higher training cost and memory requirements. Nevertheless, the online computational cost remains substantially lower than repeatedly solving the original mixed-integer non-linear optimization problem for each system state using conventional optimization methods.

6. Performance Evaluation

This section assesses the performance of the proposed JORC framework. Simulation experiments were performed to examine its effectiveness in a UAV-aided MEC environment. First, we present the simulation setup, and outline the network environment and parameter configurations. Next, we introduce the comparison baselines and explain why they are relevant for evaluating the JORC. Subsequently, we define the performance metrics used in the evaluation. Finally, we present and analyze the results, demonstrating how JORC enhances the system performance compared to existing approaches.

6.1. Simulation Setup

System setup: The simulation scenario consisted of one BS positioned at (0, 0) and two rotary-wing UAVs operating at an altitude of 100 m. The users were randomly distributed across a $2 \text{ km} \times 2 \text{ km}$ coverage area, as shown in Figure 1. Each user is served by one UAV within its communication range, whereas all UAVs stay inside the BS coverage region. The user positions are refreshed at every time step to reflect their mobility. In each timeslot, a single user generates a single computational task. The task parameters were randomly assigned within the following ranges: task size D_n^{in} was randomly set between 2 and 10 Mbit, task complexity W_n was set between 600 and 750 cycles per bit, and the maximum tolerable latency L_n^{max} was set between 1 and 7 s. The computation capacity of user devices is heterogeneous, randomly set between 0.5 GHz and 0.75 GHz. The UAVs are considered homogeneous, each with a computation capacity of 10 GHz, whereas the BS has a higher computation capacity of 50 GHz for handling more intensive workloads. Additional simulation parameters are listed in Table 3 [9,14,22,35,45,47,48].

Simulation platform: The proposed SAC-based JORC framework was developed in Python 3.7 using PyTorch 1.7. The neural networks for both policy and Q functions were designed as fully connected networks with three hidden layers, each containing 400 neurons per layer. Training was performed using the Adam optimizer with a learning rate of 1×10^{-4} , and the ReLU activation function was applied to all the layers. The discount factor was fixed at $\gamma = 0.8$, and the model was updated every 100 time slots to ensure stability and efficient learning. The target-update parameter was set to 0.005, the replay buffer size was 10^6 transitions, and the batch size was set to 64. The temperature parameter α , which controls the exploration–exploitation trade-off through entropy regularization, was initialized to $\alpha_0 = 0.2$ and adjusted automatically during training according to

(31). The target entropy $\bar{H}$ was set to the standard convention $\bar{H} = -5N$, the dimensionality of the joint action space, comprising three continuous offloading-preference scores per user, corresponding to local, assigned-UAV, and BS execution, plus one power allocation value and one CPU frequency allocation value per user, summed over the N active user devices in the scenario. The temperature parameter was updated using the same Adam optimizer and learning rate (1×10^{-4}) as the actor and critic networks. For each environment step, a single gradient update was performed [9,22,45,47,48].

The complete training procedure is summarized in Algorithm 1. During each episode, the agent interacts with the simulated environment for n_s time steps, stores the resulting transitions in the replay buffer, and updates the critic networks, actor network, and temperature parameter after each environment step using randomly sampled mini-batches. The target critic networks are then softly updated according to (32).

The hyperparameter settings for SAC were primarily adopted from the original SAC algorithm and widely used implementations [45,47,48]. Likewise, the hyperparameters of PPO, DDPG, and A3C followed their respective original publications and widely adopted implementations [9,21,22,29,48]. No extensive algorithm-specific hyperparameter tuning was performed. Instead, all algorithms were evaluated under the same problem formulation, simulation environment, training protocol, and evaluation methodology. To ensure a fair comparison, all methods employed the same backbone architecture comprising three hidden layers with 400 neurons each, while retaining their standard algorithm-specific output layers and network components. Unless otherwise required by the original algorithm, all networks were trained using the Adam optimizer with the same learning rate and evaluated using the same ten independent random seeds protocol.

To improve the statistical reliability of the evaluation, all experiments were repeated using ten independent random seeds. The reported results represent the mean performance over the ten runs, and the error bars in Figures 4–15 denote the corresponding 95% confidence intervals. In addition, paired Wilcoxon signed-rank tests were performed between JORC and each baseline across the ten random seeds for all reported metrics. The results confirm that the performance improvements of JORC are statistically significant ($p < 0.05$).

Table 3. Simulation parameters.

Parameter	Value	Parameter	Value
N	100	k	1×10^{-27}
M	2	h_0	−50 dB
H	100 m	S_u^{max}	2 GB
D_n^{in}	[2, 10] Mbit	S_b^{max}	10 GB
D_n^{out}	[0.1, 1] Mbit	$\gamma_n^L = \gamma_n^E$	0.5
W_n	[600, 750] cycles/bit	$\mathcal{W}$	50
I_n^{max}	[1, 7] s	η	0.05
E_n^{max}	20 J	r_{th}	0.5
B	20 MHz	λ	1×10^{-4}
f_m	[0.5, 0.75] GHz	n_e	2×10^3
f_u	10 GHz	n_s	1×10^3
f_b	50 GHz	ϖ	0.005
α_0	0.2	$\bar{H}$	−5 N

6.2. Comparison Baselines

To assess the effectiveness of the proposed JORC framework, we compared it to four baseline schemes. These baselines included different DRL approaches for task offloading and resource allocation in UAV-BS collaborative networks. The selected baselines were as follows:

(a) *DDPG with Caching*: Deep deterministic policy gradient (DDPG) follows an actor–critic DRL structure and is suitable for continuous action-control problems. It is widely used for optimization problems in MEC networks. This baseline applies DDPG with caching, allowing a comparison with JORC in terms of the decision making efficiency and caching effectiveness.

(b) *A3C with Caching*: Asynchronous advantage actor–critic (A3C) is another actor–critic DRL method that improves training stability by using multiple agents in parallel. This baseline evaluates whether asynchronous learning provides advantages over JORC’s SAC-based approach.

(c) *PPO with Caching*: Proximal policy optimization (PPO) is a policy-gradient DRL method that improves decision making using clipped objective functions to prevent drastic updates. This baseline assesses how a policy-based approach with caching compares to JORC’s entropy-regularized actor–critic approach.

(d) *SAC without Caching*: This baseline implements SAC without caching, indicating that task offloading and resource allocation are optimized, but no caching mechanism is applied. To ensure a fair and controlled comparison, SAC without caching baseline uses an identical neural network architecture (three hidden layers with 400 neurons each), the same hyperparameters (learning rate, discount factor, replay buffer size, and batch size), and the same training procedure as JORC. The only difference is that the caching module is entirely disabled (i.e., no cache lookup is performed before task execution), and no task-results are stored after execution. This controlled configuration separates the effect of the caching module from the remaining components of JORC.

Additionally, we evaluated the performance of JORC’s hybrid LFU-LRU caching mechanism by comparing it with three standard cache replacement strategies:

(a) *LFU (Least Frequently Used)*: stores tasks that are requested most often.

(b) *LRU (Least Recently Used)*: retains the most recently accessed tasks.

(c) *Random*: replaces cached items randomly without considering frequency or recency.

These cache replacement baselines were used to demonstrate the adaptability and effectiveness of JORC’s hybrid LFU-LRU caching policy under different request patterns.

The baseline methods were selected according to the motivation and problem formulation of this work. The primary objective of JORC is to jointly optimize task offloading, computational resource allocation, and task-result caching within a collaborative UAV-BS MEC architecture. Accordingly, our performance evaluation was designed not only to assess the overall effectiveness of the proposed framework but also to isolate the contribution of its major design components under a common problem formulation and system configuration.

Specifically, PPO, DDPG, and A3C were selected as representative actor–critic DRL algorithms because they are widely adopted for continuous optimization in MEC-related studies [28,29,32]. In contrast, value-based methods such as DQN and DDQN require substantial action discretization to handle the continuous power and CPU frequency allocation decisions considered in this work, making them less suitable for the proposed problem formulation. SAC without caching was included to quantify the contribution of the proposed task-result caching mechanism, while conventional LFU, LRU, and random replacement strategies were adopted to evaluate the proposed hybrid LFU-LRU cache replacement policy.

Although several recent UAV-assisted MEC studies investigate related optimization problems, they generally focus on different objectives, such as UAV trajectory optimization, service or content caching, user association, or federated-learning-based caching, rather than task-result caching. Consequently, directly comparing with these methods would simultaneously introduce differences in optimization objectives, caching semantics, and system architectures, making it difficult to isolate the contribution of the proposed framework. Therefore, these representative studies are discussed qualitatively in Section 2 and Table 1, while the quantitative evaluation employs controlled baselines under a common problem formulation and system configuration.

6.3. Performance Metrics

To assess the JORC framework, the following performance metrics were considered.

(a) *Average task completion latency*: This metric measures the average time required to complete a task, including transmission, processing, and caching delays. A lower average task completion latency indicates faster task execution, which is crucial for latency-sensitive services.

(b) *Average energy consumption*: This metric assesses the average energy used by a system to complete a single task, including task transmission, computations at UAVs and BS, and caching operations. Minimizing the mean energy consumption helps improve the operational efficiency of UAV-aided MEC networks, because UAVs rely on limited power resources.

(c) *Average cost*: This metric represents the average cost of processing a task, calculated as the weighted combination of latency and energy consumption. Minimizing the average cost is important for improving the overall system performance, because it balances the latency–energy tradeoff.

(d) *Successful task completion ratio (STCR)*: This metric measures the proportion of tasks completed within a given time and resource constraints. A higher successful task completion ratio reflects the system’s capability to process tasks reliably, which is necessary for latency-sensitive applications.

(e) *Offloading ratio*: This metric indicates the proportion of tasks offloaded from the local device to external resources, such as UAVs or edge servers for processing. A higher offloading ratio suggests efficient utilization of network resources and helps reduce the local device load, thereby improving overall system efficiency.

(f) *Cache Hit Ratio*: This metric represents the percentage of tasks that are retrieved from the cache without requiring recomputation. A higher cache hit ratio reduces the communication overhead and computational load, thereby improving the overall system efficiency.

These performance metrics provide a comprehensive evaluation of JORC by assessing the processing speed, energy efficiency, caching effectiveness, and overall system performance.

6.4. Results and Observations

In this subsection, we discuss our extensive simulation results and findings in detail.

6.4.1. Convergence

To examine the convergence pattern of the proposed JORC algorithm, we analyzed SAC’s learning curve against those of the three baseline DRL algorithms: PPO, DDPG, and A3C. The algorithms were trained for 2000 episodes, and Figure 3 presents the change in their performance across training episodes. As shown in Figure 3, SAC converges faster than the other algorithms. It stabilizes at approximately 670 episodes and achieves higher

average rewards, reaching approximately -230 . This rapid convergence and superior performance highlight SAC's effectiveness for the given task. In contrast, PPO and DDPG converged more slowly. Both algorithms stabilize at approximately 1000 episodes, with PPO reaching an average reward of approximately -320 and DDPG at approximately -380 . These algorithms take longer to achieve stable performance, and their rewards are lower than those of SAC. A3C shows the slowest convergence of all the algorithms. It takes approximately 1350 episodes to stabilize, with an average reward of approximately -530 . Although A3C eventually stabilized, its performance was significantly lower than those of SAC, PPO, and DDPG. Overall, SAC outperformed the other algorithms, demonstrating both faster convergence and higher average rewards, making it a stronger choice for the tasks at hand.

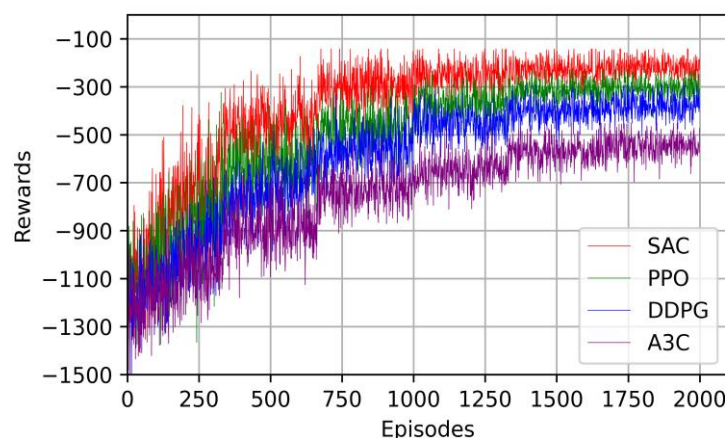


Figure 3. Convergence of SAC and three other DRL-based baseline algorithms.

6.4.2. Performance Metrics Evaluation

The performance of the proposed JORC framework is evaluated against the baseline DRL methods, PPO, DDPG, A3C (with caching), and SAC (without caching), using four performance metrics: task completion latency, energy consumption, cost, and the STCR. To evaluate the robustness and scalability of the proposed framework, experiments were conducted under different system configurations by varying the number of user devices, the number of UAVs, the UAV cache capacity, and the BS cache capacity, while keeping the remaining system parameters fixed.

Figures 4–7 and 12 present the performance of the four metrics as functions of the number of user devices, with the number of UAVs fixed at two.

Figure 4 shows the average task-completion latency as a function of the number of UD. As expected, the latency increased with the number of UD for all methods. However, JORC consistently outperformed the other methods. At 20 UD, JORC achieves a latency of approximately 1.6 s, whereas at 120 UD, the latency increases to 3.7 s. In comparison, SAC without caching showed a latency of approximately 2.4 s at 20 UD, which increased to approximately 6.4 s at 120 UD, the highest among all methods. PPO, DDPG, and A3C with caching show similar performance, with latencies ranging from 1.7 to 2 s at 20 UD and 4 to 5 s at 120 UD. These results confirm that the JORC maintains the lowest latency across all UD, demonstrating its superior performance in managing task completion times even as the number of UD increases.

Figure 5 illustrates the energy consumption as a function of the number of UD. At 20 UD, JORC consumes approximately 2.8 Joules, whereas SAC without caching requires approximately 4.1 Joules. As the number of UD increased to 120, JORC's energy consumption rose to 4.55 Joules, but it remained the most energy efficient. However, SAC

without caching increased to approximately 7.2 Joules, showing the highest energy consumption. PPO, DDPG, and A3C with caching methods exhibited moderate energy consumption, with values between 3 and 3.55 Joules at 20 UD, and 4.8 to 5.5 Joules at 120 UD. These results highlight that JORC is the most energy-efficient solution, effectively managing energy consumption even as the number of UD increases.

The average cost per task is shown in Figure 6. At 20 UD, JORC incurs a cost of approximately 0.15, which increases to 0.43 at 120 UD. By contrast, SAC without caching starts at 0.31 at 20 UD and increases to 0.8 at 120 UD, making it the most cost-inefficient method. PPO, DDPG, and A3C with caching exhibit costs ranging from 0.16 to 0.2 at 20 UD and 0.5 to 0.7 at 120 UD. This analysis indicates that JORC remains the most cost-efficient approach across different numbers of UD, effectively minimizing the cost of task processing even at the system scale.

Figure 7 depicts STCR as a function of the number of UD. At 20 UD, JORC achieved an STCR of 0.98, where SAC without caching reached only 0.92. As the number of UD increased, the STCR decreased for all methods. At 120 UD, JORC maintained a high STCR of 0.88, whereas SAC without caching dropped to 0.7, which was the lowest among all methods. PPO, DDPG, and A3C with caching show moderate performance, with STCRs ranging from 0.78 to 0.85 at 120 UD.

Next, the robustness and scalability of the proposed framework are evaluated by varying the number of UAVs while fixing the number of user devices at 100, as shown in Figures 8–11 and 13.

Figure 8 illustrates the variation in average task completion latency as the number of UAVs increases from one to eight. As expected, latency decreases with the addition of UAVs, benefiting from more efficient task distribution and the expanded caching capacity. Specifically, with a single UAV, JORC achieves a latency of approximately 3.4 s, outperforming PPO, DDPG, A3C (with caching), and SAC (without caching). As the number of UAVs increases to five, JORC's latency significantly reduces to around 1.8 s, maintaining the lowest latency among all methods. However, beyond five UAVs, the improvements in latency become marginal, with JORC stabilizing at approximately 1.7 s from 6 to 8 UAVs. A similar trend of diminishing returns is observed for the other methods, although their latencies remain consistently higher than JORC's. In contrast, SAC without caching consistently shows higher latency, with a slight increase in latency beyond seven UAVs, likely due to suboptimal task distribution in the absence of caching mechanisms. These findings suggest that, while increasing the number of UAVs enhances system performance, the benefits saturate beyond a certain point, suggesting an optimal range for UAV deployment.

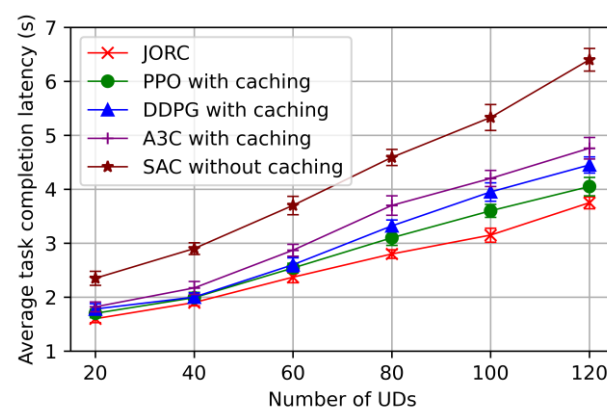


Figure 4. Average task completion latency vs. the number of UD.

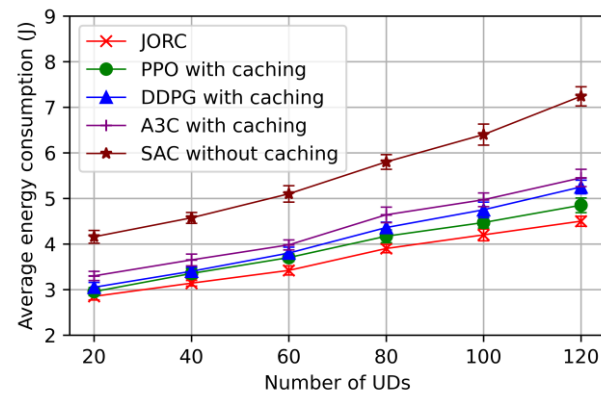


Figure 5. Average energy consumption vs. the number of UDs.

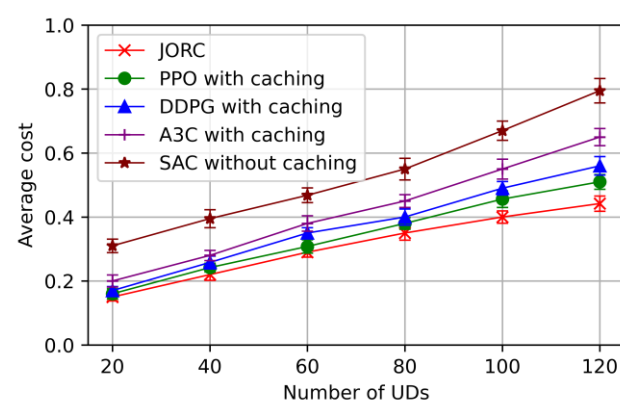


Figure 6. Average cost vs. the number of UDs.

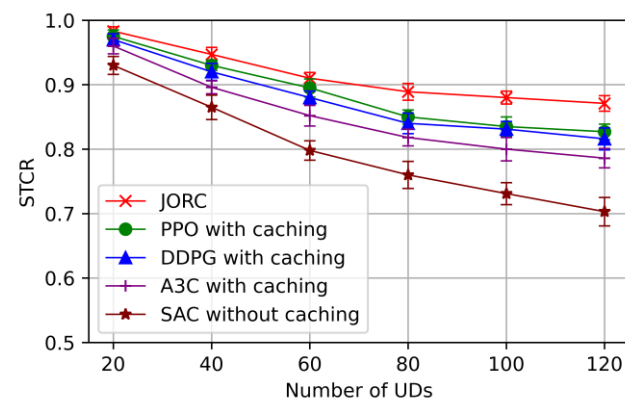


Figure 7. Successful task completion ratio vs. the number of UDs.

Figure 9 depicts the relationship between the number of UAVs and average energy consumption. As the number of UAVs increases, energy consumption generally decreases due to the enhanced task distribution and greater cache availability, reducing unnecessary transmissions, and computational load. JORC consistently exhibits the lowest energy consumption across all methods. For instance, with a single UAV, JORC consumes about 4.9 J, which decreases to approximately 3.9 J when five UAVs are deployed. Beyond this point, the improvement in energy efficiency becomes negligible, with consumption stabilizing, indicating a saturation threshold. A similar leveling off is seen in PPO, DDPG, and A3C with caching, though these methods consistently consume more energy than JORC at every UAV count. In contrast, SAC without caching shows the highest energy consumption, with a gradual increase after four UAVs, likely due to inefficient task offloading

in the absence of caching. These results highlight that, while increasing the number of UAVs initially enhances energy efficiency, the benefit stagnates, suggesting that five UAVs is an optimal number for minimizing energy consumption in the given condition.

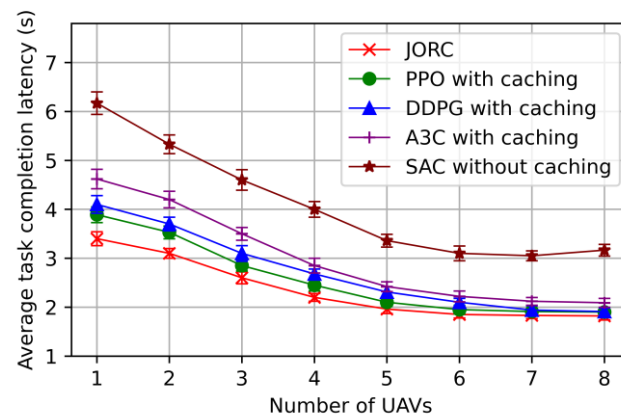


Figure 8. Average task completion latency vs. the number of UAVs.

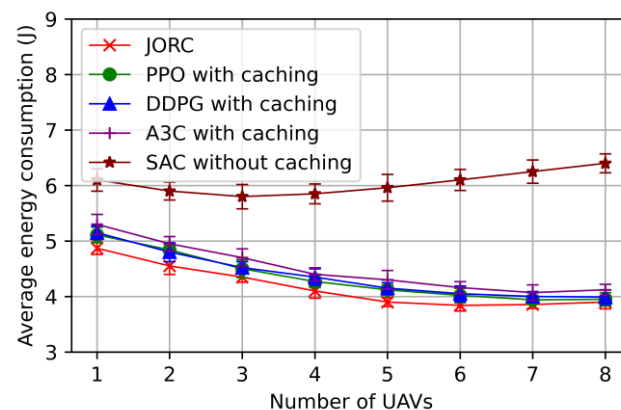


Figure 9. Average energy consumption vs. the number of UAVs.

Figure 10 illustrates the average task completion cost as the number of UAVs increases from one to eight. The cost, representing a weighted combination of latency and energy consumption, decreases steadily for all methods as more UAVs are deployed. JORC consistently achieves the lowest average cost, reducing from approximately 0.46 with a single UAV to around 0.17 with six UAVs. Beyond this point, the cost stabilizes, showing diminishing returns. PPO, DDPG, and A3C with caching also exhibit a downward trend, but their costs remain slightly higher than JORC's, stabilizing between 0.18 and 0.2 at eight UAVs. In contrast, SAC without caching starts with the highest cost, around 0.75 at a single UAV, and shows minimal improvement after six UAVs. These findings suggest that, while adding UAVs initially improves cost efficiency, the benefit levels off after six UAVs, indicating that six UAVs is an optimal point for efficient deployment in the given condition.

Figure 11 depicts the STCR as a function of the number of UAVs. As the number of UAVs increases, the STCR improves for all methods, reflecting more efficient task management and a reduced system overload. JORC consistently achieves the highest STCR across all UAV counts, starting at approximately 0.84 with a single UAV and steadily rising to around 0.965 with five UAVs. Beyond five UAVs, the improvement becomes minimal, with STCR stabilizing near 0.97 at eight UAVs. PPO, DDPG, and A3C with caching also show steady improvement, reaching STCR values between 0.93 and 0.96 at eight UAVs, but remain slightly lower than JORC. In contrast, SAC without caching starts with

the lowest STCR of about 0.70 and only reaches 0.86 at eight UAVs, indicating inefficiency without caching. These results suggest that, while increasing the number of UAVs improves the task success rate, the gains level off beyond five UAVs, highlighting an optimal point for maximizing STCR in the given condition.

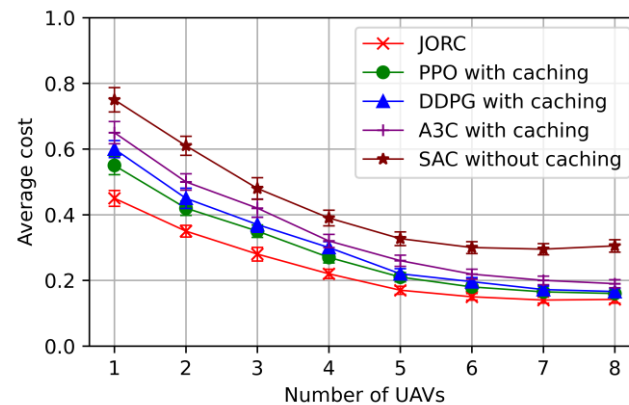


Figure 10. Average cost vs. the number of UAVs.

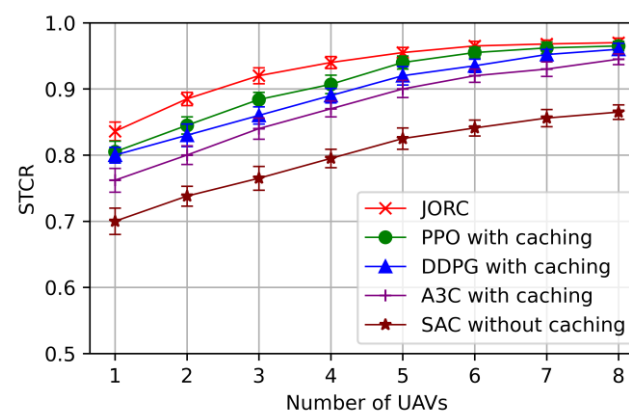


Figure 11. Successful task completion ratio vs. the number of UAVs.

Figures 12 and 13 illustrate the offloading ratio vs. number of UD and UAVs. As shown in Figure 12, as the number of UD increases from 20 to 120 (where the number of UAVs is fixed at 2), the offloading ratio tends to remain relatively stable for all methods, with minor fluctuations. JORC achieved the highest offloading ratio across all UD. The offloading ratio for JORC is nearly identical across the whole range, indicating its robustness in task offloading regardless of the load from the UD. PPO, DDPG, and A3C, all using caching mechanisms, follow closely behind JORC, with offloading ratios ranging from 0.4 to 0.6 across the UD. Although these methods show a slight decrease as the number of UD increases, their performance remains strong in comparison with that of SAC. SAC without caching exhibited the highest offloading ratio at all UD, peaking at approximately 0.6 for smaller numbers of UD and gradually dropping below 0.6 as the load increased. The significant gap between SAC without caching and the other methods highlights the importance of caching in improving the task offloading efficiency.

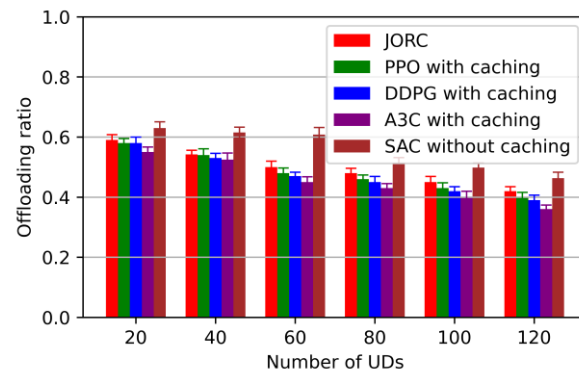


Figure 12. Offloading ratio vs. the number of UD's.

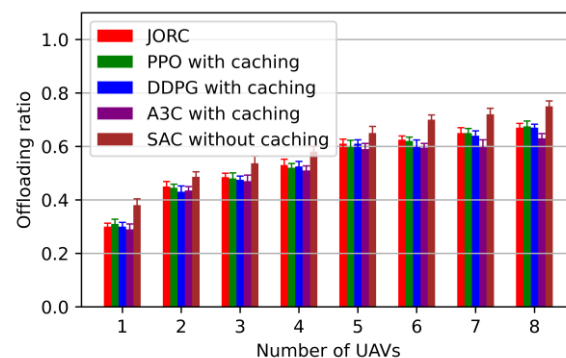


Figure 13. Offloading ratio vs. the number of UAV's.

Figure 13 depicts the offloading ratio as a function of the number of UAVs, ranging from one to eight (where the number of UD's is fixed at 100). As expected, the offloading ratio increases with the number of UAVs due to both the enhanced computing capacity and the improved task distribution. JORC, PPO, DDPG, and A3C (all with caching) show similar trends, with the offloading ratio steadily improving as the number of UAVs increases. However, SAC without caching consistently shows the highest offloading ratio across all UAV counts. This is because SAC without caching lacks a caching policy, resulting in no cache hits and, thus, every task must either be computed locally or offloaded to the UAV or BS. In contrast, methods with caching (such as JORC, PPO, DDPG, and A3C) show lower offloading ratio compared to SAC without caching, highlighting the benefit of the caching mechanism in reducing communication and computation overhead. These results emphasize that incorporating caching strategies results in more efficient task management by decreasing the need for redundant computations and offloading, thus reducing the overall system load.

Finally, the impact of caching resources is evaluated by varying the UAV and BS cache capacities. Figures 14 and 15 present the cache hit ratio versus the UAV and BS cache capacities, respectively, for a network with two UAVs, one BS, and 100 UD's.

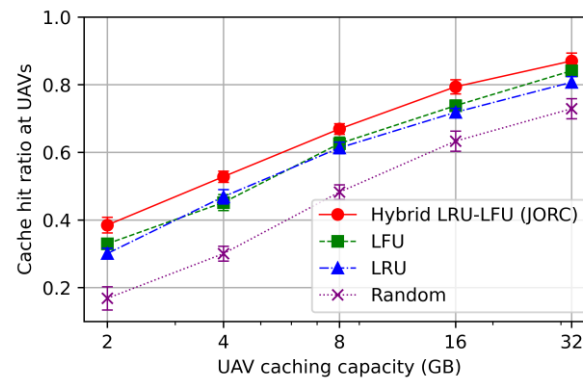


Figure 14. Cache hit ratio vs. UAV caching capacity.

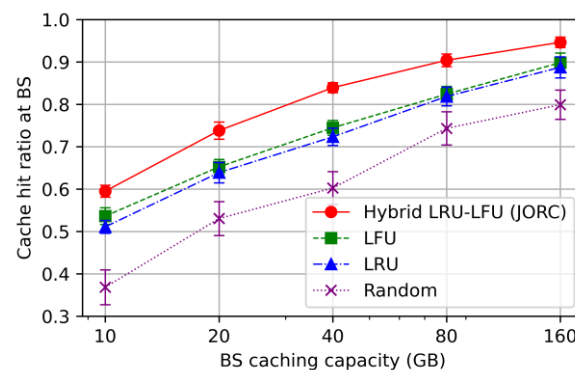


Figure 15. Cache hit ratio vs. BS caching capacity.

Figure 14 shows the cache hit ratio of the UAVs as a function of caching capacity. The figure illustrates that the cache hit ratio improves as the caching capacity increases for all methods. JORC consistently achieves the highest cache hit ratio, starting at 0.4 for a 2 GB caching capacity and reaching 0.9 at a 32 GB caching capacity. This clear upward trend demonstrates the effectiveness of the Hybrid LRU-LFU (JORC) caching strategy in improving the cache performance as the available storage capacity increases. The LFU and LRU caching mechanisms showed a similar upward trend, but with lower cache hit ratios. LFU increases from 0.33 to approximately 0.82 as caching capacity grows, while LRU progresses from 0.3 to approximately 0.8 at 32 GB. Random caching exhibited the lowest cache hit ratio, starting at 0.18 and only reaching 0.72 at the highest caching capacity. These results emphasize the superior performance of JORC, which significantly outperformed the other methods, particularly as the caching capacity increased.

Figure 15 shows the cache hit ratios at the BS for various caching capacities. Similarly to the UAV results, the cache hit ratio improved with an increase in caching capacity. JORC once again leads with the highest hit ratio, which starts at 0.6 for 10 GB and reaches approximately 0.95 at 160 GB. LFU and LRU show similar but lower trends, with LFU ranging from 0.53 to 0.9 and LRU ranging from 0.51 to 0.89 at the highest caching capacity. Random caching showed the least improvement, from 0.38 to 0.8, demonstrating its inefficiency. Figures 14 and 15 show that the Hybrid LRU-LFU (JORC) consistently delivered the highest cache hit ratio for both UAVs and the BS, and the performance of all methods improved with an increase in caching capacity.

7. Discussion

This section discusses the key findings of the proposed JORC framework, outlines the assumptions and limitations of the current system model, and identifies several promising directions for future research.

7.1. Discussion of Experimental Results

The superior performance of JORC can be attributed to two complementary mechanisms. First, compared with PPO, DDPG, and A3C (all equipped with the same caching mechanism), JORC benefits from SAC's entropy-regularized off-policy learning, which achieves faster and more stable policy convergence as demonstrated by the convergence results in Section 6.4.1. Consequently, SAC learns more effective task offloading and resource allocation policies than the other DRL algorithms under identical caching support, resulting in consistently lower latency, energy consumption, and overall system cost across the evaluated scenarios. Second, compared with SAC without caching, the additional performance gain originates from the proposed hybrid LFU-LRU task-result caching mechanism. By reusing previously computed task-results whenever cache hits occur, redundant task transmission and computation are substantially reduced, thereby lowering latency, energy consumption, and overall system cost while improving cache utilization.

The performance gains are not uniform across all operating conditions but become more pronounced as the system load increases relative to the available resources. As the number of UDs increases from 20 to 120 (Figures 4–7), the performance gap between JORC and SAC without caching gradually widens because a larger user population increases the probability of repeated task requests, enabling the caching mechanism to eliminate an increasing amount of redundant computation. Likewise, the advantage of JORC over the other DRL baselines is most evident when only a small number of UAVs are available (Figures 8–11), where communication and computing resources are more constrained and efficient resource management becomes increasingly important. As the number of UAVs increases, additional computing resources reduce resource contention for all methods, causing the performance differences to gradually decrease and eventually approach saturation.

Overall, the experimental results demonstrate that the major components of JORC provide complementary benefits. Comparisons with PPO, DDPG, and A3C under identical caching support validate the effectiveness of SAC for joint task offloading and computational resource allocation, while the comparison with SAC without caching quantifies the additional benefit of the proposed task-result caching mechanism. Furthermore, comparisons with LFU, LRU, and random replacement strategies confirm the effectiveness of the proposed hybrid LFU-LRU cache replacement policy. The robustness of these findings is supported by experiments conducted over ten independent random seeds, with 95% confidence intervals reported in Figures 4–15 and paired Wilcoxon signed-rank tests confirming that the performance improvements of JORC are statistically significant ($p < 0.05$).

7.2. Model Assumptions and Limitations

To maintain analytical tractability, the proposed framework adopts several simplifying assumptions whose practical implications should be acknowledged.

First, the UAV-assisted MEC architecture models UAVs as lightweight aerial MEC nodes that provide localized and temporary computing support, while the terrestrial BS performs more computationally intensive processing. This collaborative architecture is consistent with the UAV-enabled MEC paradigm reported in the literature, where UAV-

mounted MEC servers complement terrestrial infrastructure in scenarios such as communication hotspots, disaster recovery, and areas without reliable terrestrial MEC coverage [49]. The proposed framework explicitly models the communication and computation energy associated with task execution, whereas UAV propulsion energy, battery discharge dynamics, and flight-endurance constraints are beyond the scope of the present study.

Second, the proposed task-result caching mechanism assumes deterministic computation tasks, where the repeated requests with identical task inputs produce identical outputs. This assumption enables efficient task-result reuse and facilitates the joint optimization of task offloading, resource allocation, and task-result caching. However, in practical MEC systems, the cached results may become outdated because of evolving task inputs or changes in the underlying data, requiring cache invalidation and consistency management. Moreover, task identification is assumed to be deterministic, such that cache reuse is triggered only when the same task with identical input is requested. Applications involving continuously varying or similar inputs may therefore require more advanced techniques, such as content-based hashing or similarity-aware matching, which are beyond the scope of the present work.

Third, the communication model considers only large-scale path loss under LoS-dominant and quasi-static assumptions, allowing the analysis to focus on the proposed joint optimization framework. Consequently, small-scale fading and its impact on transmission rates and task offloading decisions are not considered. In addition, the BS and each UAV are assumed to employ single-antenna transmission, consistent with the scalar channel gains in (1) and (2), and therefore do not capture multi-antenna transmission or beamforming techniques. Furthermore, the downlink transmission latency and energy are assumed to be negligible because the output data size is considerably smaller than the input task size and the downlink generally provides a higher transmission rate. Cache lookup is also assumed to be instantaneous and error-free, without considering lookup delay or cache-management overhead. These assumptions are consistent with commonly adopted abstractions in related UAV-assisted MEC studies [14,15,44], and enable the study to focus on the joint optimization problem.

Finally, all the reported results are obtained through simulation and therefore primarily demonstrate the algorithmic behavior of JORC under controlled UAV-assisted MEC conditions. Although simulation enables systematic comparison under identical network settings, it cannot fully capture practical factors such as hardware processing latency, signaling overhead, imperfect state information, cache synchronization, wireless-channel uncertainty, and UAV operational constraints. Furthermore, although our comparisons with the controlled baselines adopted in this work are designed to both isolate the contribution of each major design component and evaluate the overall effectiveness of the integrated JORC framework under a common problem formulation and system configuration, the quantitative evaluation does not include direct comparisons with recent state-of-the-art UAV-assisted MEC optimization frameworks. This represents a limitation of the present experimental evaluation. Consequently, the reported results should be interpreted as evidence of the potential effectiveness of JORC rather than as direct proof of deployment performance.

7.3. Future Research Directions

Several promising directions may further extend the proposed JORC framework. On the caching side, future work will investigate more practical task-result caching strategies incorporating cache invalidation, distributed cache-consistency management, evolving task inputs, and similarity-aware or approximate cache matching, enabling semantically similar requests to benefit from cached computation results.

The proposed UAV-assisted MEC framework can also be extended by jointly considering communication, computation, and propulsion energy, together with battery dynamics, flight-endurance constraints, and UAV trajectory optimization, to provide a more realistic representation of UAV operation.

The communication model may be further enhanced by incorporating small-scale fading, shadowing, mobility-induced channel variations, and multi-antenna transmission with beamforming, thereby enabling the joint optimization of beamforming, task offloading, computational resource allocation, and task-result caching under more realistic wireless environments.

Finally, an important direction for future research is the practical validation of the proposed framework through comprehensive runtime benchmarking, scalability analysis under larger network deployments, hardware-in-the-loop experiments, UAV-assisted MEC testbeds, and ultimately real-world deployments. Such studies will provide further insight into the practical applicability, robustness, and deployment performance of JORC under realistic operating conditions.

8. Conclusions

This study introduced the JORC framework for improving task execution in UAV-aided MEC networks, specifically for latency-sensitive applications. The framework jointly optimizes task offloading and resource allocation using a SAC-based DRL approach, while incorporating a hybrid LFU-LRU caching mechanism to enhance system efficiency. The proposed scheme effectively improves overall system performance by reducing task execution delays, optimizing resource distribution, and minimizing redundant computations. The simulation results demonstrated that the proposed JORC achieves better performance than existing methods in terms of latency reduction, energy efficiency, cache utilization, and overall cost minimization, making it a practical solution for real-time services, such as AR/VR, autonomous driving, and IoT-based applications. The combination of intelligent offloading and resource allocation and adaptive caching enhances system performance under dynamic network conditions in UAV-aided MEC systems.

Author Contributions: T.B. developed the JORC framework, implemented the simulation model, conducted the experiments, and analyzed the results. S.M. supervised the research, provided guidance in refining the proposed framework, and contributed to the interpretation of the simulation results. The manuscript was drafted by T.B. and critically reviewed and revised by S.M. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by a research fund from Chosun University, 2025.

Data Availability Statement: The data presented in this study are available upon reasonable request, subject to approval by the funding organization and applicable institutional and copyright requirements. The implementation and simulation framework developed for this study is being considered for public release through a public repository (e.g., GitHub), subject to approval by the funding organization and applicable institutional and copyright requirements.

Acknowledgments: The authors thank the editor and anonymous referees for their comments, which helped to improve the quality of this paper. This article is based on the first author T. Baidya's master's thesis supervised by the second author S. Moh [50].

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Ali, M.; Naeem, F.; Kaddoum, G.; Hossain, E. Metaverse Communications, Networking, Security, and Applications: Research Issues, State-of-the-Art, and Future Directions. *IEEE Commun. Surv. Tutor.* **2024**, *26*, 1238–1278. <https://doi.org/10.1109/COMST.2023.3347172>.
2. Yoo, S.; Jeong, S.; Kim, J.; Kang, J. Cache-Assisted Mobile-Edge Computing Over Space–Air–Ground Integrated Networks for Extended Reality Applications. *IEEE Internet Things J.* **2024**, *11*, 18306–18319. <https://doi.org/10.1109/JIOT.2024.3361907>.
3. Chen, C.-L.; Bhargava, B.; Aggarwal, V.; Tonshal, B.; Gopal, A. A Hybrid Deep Reinforcement Learning Approach for Jointly Optimizing Offloading and Resource Management in Vehicular Networks. *IEEE Trans. Veh. Technol.* **2024**, *73*, 2456–2467. <https://doi.org/10.1109/TVT.2023.3312340>.
4. Shen, Q.; Hu, B.-J.; Xia, E. Dependency-Aware Task Offloading and Service Caching in Vehicular Edge Computing. *IEEE Trans. Veh. Technol.* **2022**, *71*, 13182–13197. <https://doi.org/10.1109/TVT.2022.3196544>.
5. Zhao, M.; Zhang, R.; He, Z.; Li, K. Joint Optimization of Trajectory, Offloading, Caching, and Migration for UAV-Assisted MEC. *IEEE Trans. Mob. Comput.* **2025**, *24*, 1981–1998. <https://doi.org/10.1109/TMC.2024.3486995>.
6. Li, Y.; Huynh, D.V.; Nguyen, V.-L.; Ha, D.-B.; Zepernick, H.-J.; Duong, T.Q. Multiagent UAV-Aided URLLC Mobile Edge Computing Systems: A Joint Communication and Computation Optimization Approach. *IEEE Syst. J.* **2024**, *18*, 1828–1838. <https://doi.org/10.1109/JSYST.2024.3426096>.
7. Poudel, S.; Moh, S. Task Assignment Algorithms for Unmanned Aerial Vehicle Networks: A Comprehensive Survey. *Veh. Commun.* **2022**, *35*, 100469. <https://doi.org/10.1016/j.vehcom.2022.100469>.
8. Huda, S.M.A.; Moh, S. Survey on Computation Offloading in UAV-Enabled Mobile Edge Computing. *J. Netw. Comput. Appl.* **2022**, *201*, 103341. <https://doi.org/10.1016/j.jnca.2022.103341>.
9. Raivi, A.M.; Moh, S. JDACO: Joint Data Aggregation and Computation Offloading in UAV-Enabled Internet of Things for Post-Disaster Scenarios. *IEEE Internet Things J.* **2024**, *11*, 16529–16544. <https://doi.org/10.1109/JIOT.2024.3354950>.
10. Nabi, A.; Baidya, T.; Moh, S. Comprehensive Survey on Reinforcement Learning-Based Task Offloading Techniques in Aerial Edge Computing. *Internet Things* **2024**, *28*, 101342. <https://doi.org/10.1016/j.iot.2024.101342>.
11. Yan, H.; Li, H.; Xu, X.; Bilal, M. UAV-Enhanced Service Caching for IoT Systems in Extreme Environments. *IEEE Internet Things J.* **2024**, *11*, 26741–26750. <https://doi.org/10.1109/JIOT.2023.3288200>.
12. Baidya, T.; Moh, S. Comprehensive Survey on Resource Allocation for Edge-Computing-Enabled Metaverse. *Comput. Sci. Rev.* **2024**, *54*, 100680. <https://doi.org/10.1016/j.cosrev.2024.100680>.
13. Yang, S.; Liu, J.; Zhang, F.; Li, F.; Chen, X.; Fu, X. Caching-Enabled Computation Offloading in Multi-Region MEC Network via Deep Reinforcement Learning. *IEEE Internet Things J.* **2022**, *9*, 21086–21098. <https://doi.org/10.1109/JIOT.2022.3176289>.
14. Li, Y.; Zhu, X.; Li, N.; Wang, L.; Chen, Y.; Yang, F.; Zhai, L. Collaborative Content Caching and Task Offloading in Multi-Access Edge Computing. *IEEE Trans. Veh. Technol.* **2023**, *72*, 5367–5372. <https://doi.org/10.1109/TVT.2022.3222596>.
15. Feng, H.; Guo, S.; Yang, L.; Yang, Y. Collaborative Data Caching and Computation Offloading for Multi-Service Mobile Edge Computing. *IEEE Trans. Veh. Technol.* **2021**, *70*, 9408–9422. <https://doi.org/10.1109/TVT.2021.3099303>.
16. Chen, J.; Yang, P.; Ren, S.; Zhao, Z.; Cao, X.; Wu, D. Enhancing AIoT Device Association with Task Offloading in Aerial MEC Networks. *IEEE Internet Things J.* **2024**, *11*, 174–187. <https://doi.org/10.1109/JIOT.2023.3300011>.
17. Yuan, H.; Wang, M.; Bi, J.; Shi, S.; Yang, J.; Zhang, J.; Zhou, M.; Buyya, R. Cost-Efficient Task Offloading in Mobile Edge Computing With Layered Unmanned Aerial Vehicles. *IEEE Internet Things J.* **2024**, *11*, 30496–30509. <https://doi.org/10.1109/JIOT.2024.3408216>.
18. Wang, Q.; Ma, X.; Jiang, Z.; Zhu, Y. Joint Optimization of Task Offloading, Resource Allocation, and Trajectory Design in Cooperative Multi-UAV MEC Networks. *IEEE Internet Things J.* **2026**, *13*, 24413–24425. <https://doi.org/10.1109/JIOT.2026.3673398>.
19. Liu, X.; Liu, S.; Zhu, J.; An, Y.; Mu, C. Service Capacity Maximization in Multi-UAV MEC: Deployment, Offloading, and Resource Allocation. *IEEE Trans. Veh. Technol.* **2026**, *75*, 14711–14724. <https://doi.org/10.1109/TVT.2026.3660629>.
20. Liu, B.; Liu, C.; Peng, M. Computation Offloading and Resource Allocation in Unmanned Aerial Vehicle Networks. *IEEE Trans. Veh. Technol.* **2023**, *72*, 4981–4995. <https://doi.org/10.1109/TVT.2022.3222907>.
21. Kang, H.; Chang, X.; Mišić, J.; Mišić, V.B.; Fan, J.; Liu, Y. Cooperative UAV Resource Allocation and Task Offloading in Hierarchical Aerial Computing Systems: A MAPPO-Based Approach. *IEEE Internet Things J.* **2023**, *10*, 10497–10509. <https://doi.org/10.1109/JIOT.2023.3240173>.
22. Morshed Alam, M.; Moh, S. Joint Optimization of Trajectory Control, Task Offloading, and Resource Allocation in Air-Ground Integrated Networks. *IEEE Internet Things J.* **2024**, *11*, 24273–24288. <https://doi.org/10.1109/JIOT.2024.3390168>.

23. Qin, P.; Wu, X.; Fu, M. Latency Minimization Resource Allocation and Trajectory Optimization for UAV-Assisted Cache-Computing Network with Energy Recharging. *IEEE Trans. Commun.* **2025**, *73*, 5715–5728. <https://doi.org/10.1109/TCOMM.2025.3534587>.
24. Yang, L.; Zheng, J.; Zhang, Y. A Deep Reinforcement Learning-Based Service Caching and Computation Offloading Algorithm for Multi-UAV-Enabled MEC Systems. *IEEE Trans. Cogn. Commun. Netw.* **2026**, *12*, 6658–6669. <https://doi.org/10.1109/TCCN.2026.3670152>.
25. Xiao, T.; Lu, C.; Xu, X.; Zhang, Y. Task Offloading in Hierarchical UAV-Enabled MEC Network: A Trust Region Policy Optimization-Based Approach. *IEEE Trans. Transp. Electr.* **2026**, *12*, 3999–4010. <https://doi.org/10.1109/TTE.2026.3658558>.
26. Huang, J.; Zhang, M.; Wan, J.; Chen, Y.; Zhang, N. Joint Data Caching and Computation Offloading in UAV-Assisted Internet of Vehicles via Federated Deep Reinforcement Learning. *IEEE Trans. Veh. Technol.* **2024**, *73*, 17644–17656. <https://doi.org/10.1109/TVT.2024.3429507>.
27. Sun, C.; Li, X.; Wang, C.; He, Q.; Wang, X.; Leung, V.C.M. Hierarchical Deep Reinforcement Learning for Joint Service Caching and Computation Offloading in Mobile Edge-Cloud Computing. *IEEE Trans. Serv. Comput.* **2024**, *17*, 1548–1564. <https://doi.org/10.1109/TSC.2024.3355937>.
28. Yang, Q.; Yoo, S. Adaptive UAV-Aided Fair Task Offloading with Fuzzy Caching for Vehicular Edge Computing via Hierarchical Deep Reinforcement Learning. *IEEE Trans. Veh. Technol.* **2025**, *75*, 11558–11574. <https://doi.org/10.1109/TVT.2025.3647844>.
29. Bai, J.; Luo, J.; Chen, Y. The DDPG-Based Joint Optimization of Task Offloading and Content Caching in UAV-Assisted IoV. *IEEE Internet Things J.* **2025**, *12*, 40330–40346. <https://doi.org/10.1109/JIOT.2025.3588858>.
30. Li, C.; Jiang, K.; Zhang, Z. Joint Service Caching and Computation Offloading Scheme With 3D UAV Deployment for ICVs In UAV-Assisted VEC. *IEEE Trans. Commun.* **2025**, *73*, 10886–10899. <https://doi.org/10.1109/TCOMM.2025.3581093>.
31. Wang, X.; He, C.; Liu, X. Generative AI-Based Dependency-Aware Task Offloading and Resource Allocation for UAV-Assisted IoV. *IEEE Open J. Commun. Soc.* **2025**, *6*, 3932–3949. <https://doi.org/10.1109/OJCOMS.2025.3562720>.
32. Qin, P.; Li, M.; Wu, K.; Fu, Y. Cost Minimization Resource Allocation with Service Instance Caching and Task Migration for UAV Mobile Edge Computing. *IEEE Trans. Netw. Sci. Eng.* **2026**, *13*, 1738–1753. <https://doi.org/10.1109/TNSE.2025.3600480>.
33. Zhang, Y.; Na, Z.; Lin, Y.; Nallanathan, A. Joint Service Caching and Task Offloading for Multi UAV-Assisted Offshore Edge Computing Networks. *IEEE Trans. Veh. Technol.* **2025**, *74*, 19667–19680. <https://doi.org/10.1109/TVT.2025.3589880>.
34. Zhou, R.; Huang, Y.; Wang, Y.; Jiao, L.; Tan, H.; Zhang, R.; Wu, L. User Preference Oriented Service Caching and Task Offloading for UAV-Assisted MEC Networks. *IEEE Trans. Serv. Comput.* **2025**, *18*, 1097–1109. <https://doi.org/10.1109/TSC.2025.3536319>.
35. Chen, S.; Rui, L.; Gao, Z.; Li, W.; Qiu, X. Cache-Assisted Collaborative Task Offloading and Resource Allocation Strategy: A Metareinforcement Learning Approach. *IEEE Internet Things J.* **2022**, *9*, 19823–19842. <https://doi.org/10.1109/JIOT.2022.3168885>.
36. Wang, L.; Zhang, G. Joint Service Caching, Resource Allocation and Computation Offloading in Three-Tier Cooperative Mobile Edge Computing System. *IEEE Trans. Netw. Sci. Eng.* **2023**, *10*, 3343–3353. <https://doi.org/10.1109/TNSE.2023.3259030>.
37. Qin, P.; Fu, M.; Fu, Y.; Ding, R.; Zhao, X. Collaborative Edge Computing and Program Caching with Routing Plan in C-NOMA-Enabled Space-Air-Ground Network. *IEEE Trans. Wirel. Commun.* **2024**, *23*, 18302–18315. <https://doi.org/10.1109/TWC.2024.3464610>.
38. Lin, N.; Han, X.; Hawbani, A.; Sun, Y. Deep Reinforcement Learning-Based Dual-Timescale Service Caching and Computation Offloading for Multi-UAV Assisted MEC Systems. *IEEE Trans. Netw. Serv. Manag.* **2025**, *22*, 605–617. <https://doi.org/10.1109/TNSM.2024.3468312>.
39. Gao, X.; Zhai, L. Service Experience Oriented Cooperative Computing in Cache-Enabled UAVs Assisted MEC Networks. *IEEE Trans. Mob. Comput.* **2024**, *23*, 9721–9736. <https://doi.org/10.1109/TMC.2024.3366944>.
40. Fan, C.; Zhan, J.; Wang, J.; Mao, S. Multi-Task Parallel Execution-Oriented Content Caching, Computation Offloading and Channel Allocation in UAV-Assisted MEC Network. *IEEE Trans. Mob. Comput.* **2026**, *25*, 12591–12607. <https://doi.org/10.1109/TMC.2026.3674329>.
41. Barman, D.C.; Chakraborty, S.; Hazra, A. Joint Service Caching, Task Offloading, and Service Delivery in Mobile Edge Computing for Latency Critical UAV Networks. *IEEE Trans. Comput. Soc. Syst.* **2025**, *13*, 3717–3729. <https://doi.org/10.1109/TCSS.2025.3643164>.
42. Yang, Y.; Gong, Y. Intelligent-Reflecting-Surface-Aided Mobile Edge Computing with Binary Offloading: Energy Minimization for IoT Devices. *IEEE Internet Things J.* **2022**, *9*, 12973–12983. <https://doi.org/10.1109/JIOT.2022.3173027>.
43. Chen, Y.; Wen, M.; Basar, E.; Wu, Y. Exploiting Reconfigurable Intelligent Surfaces in Edge Caching: Joint Hybrid Beamforming and Content Placement Optimization. *IEEE Trans. Wirel. Commun.* **2021**, *20*, 7799–7812. <https://doi.org/10.1109/TWC.2021.3087912>.

44. Zheng, G.; Xu, C.; Wen, M.; Zhao, X. Service Caching Based Aerial Cooperative Computing and Resource Allocation in Multi-UAV Enabled MEC Systems. *IEEE Trans. Veh. Technol.* **2022**, *71*, 10934–10947. <https://doi.org/10.1109/TVT.2022.3183577>.
45. Haarnoja, T.; Zhou, A.; Hartikainen, K.; Tucker, G.; Ha, S.; Tan, J.; Kumar, V.; Zhu, H.; Gupta, A.; Abbeel, P.; et al. Soft Actor-Critic Algorithms and Applications. *arXiv* **2018**, arXiv:1812.05905.
46. Haarnoja, T.; Zhou, A.; Abbeel, P.; Levine, S. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018; Volume 5, pp. 2976–2989.
47. Nabi, A.; Moh, S. Joint Offloading Decision, User Association, and Resource Allocation in Hierarchical Aerial Computing: Collaboration of UAVs and HAP. *IEEE Trans. Mob. Comput.* **2025**, *24*, 7267–7282. <https://doi.org/10.1109/TMC.2025.3548668>.
48. Wang, C.; Li, R.; Wang, X.; Taleb, T.; Guo, S.; Sun, Y.; Leung, V.C.M. Heterogeneous Edge Caching Based on Actor-Critic Learning with Attention Mechanism Aiding. *IEEE Trans. Netw. Sci. Eng.* **2023**, *10*, 3409–3420. <https://doi.org/10.1109/TNSE.2023.3260882>.
49. Zhou, F.; Hu, R.Q.; Li, Z.; Wang, Y. Mobile Edge Computing in Unmanned Aerial Vehicle Networks. *IEEE Wirel. Commun.* **2020**, *27*, 140–146. <https://doi.org/10.1109/MWC.001.1800594>.
50. Baidya, T. Joint Offloading, Resource Allocation, and Caching for UAV-Assisted Edge Computing Systems. Master's Thesis, Department of Computer Engineering, Chosun University, Gwangju, Republic of Korea, 2025.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.