IEEE Open Journal of Vehicular Technology

Received 24 July 2025; accepted 9 August 2025. Date of publication 11 August 2025;
date of current version 27 August 2025. The review of this article was coordinated by Editor Jianbo Du.

Digital Object Identifier 10.1109/OJVT.2025.3598154

Energy Efficient and Resilient Task Offloading in UAV-Assisted MEC Systems

MOHAMED EL-EMARY [1], DIALA NABOULSI [1] (Member, IEEE), AND RAZVAN STANICA [2]

[1]Department of Software Engineering and IT, École de Technologie Supérieure, Montréal, QC H3C1K3, Canada
[2]INSA Lyon, Inria, CITI, F-69621 Villeurbanne, France

CORRESPONDING AUTHOR: MOHAMED EL-EMARY (e-mail: mohamed-ibrahim-mahmoud.el-emary.1@ens.etsmtl.ca).

ABSTRACT Unmanned aerial vehicle (UAV)-assisted Mobile Edge Computing (MEC) presents a critical trade-off between minimizing user equipment (UE) energy consumption and ensuring high task execution reliability, especially for mission-critical applications.While many frameworks focus on either energy efficiency or resiliency, few address both objectives simultaneously with a structured redundancy model. To bridge this gap, this paper proposes a novel reinforcement learning (RL)-based framework that intelligently distributes computational tasks among UAVs and base stations (BSs). We introduce an $(h + 1)$-server permutation strategy that redundantly assigns tasks to multiple edge servers, guaranteeing execution continuity even under partial system failures. An RL agent optimizes the offloading process by leveraging network state information to balance energy consumption with system robustness. Extensive simulations demonstrate the superiority of our approach over state-of-the-art benchmarks. Notably, our proposed framework sustains average UE energy levels above 75% under high user densities, exceeds 95% efficiency with more base stations, and maintains over 90% energy retention when 20 or more UAVs are deployed. Even under high computational loads, it preserves more than 50% of UE energy, outperforming all benchmarks by a significant margin—especially for mid-range task sizes where it leads by over 15–20% in energy efficiency. These findings highlight the potential of our framework to support energy-efficient and failure-resilient services for next-generation wireless networks.

INDEX TERMS Unmanned aerial vehicles, mobile edge computing, task offloading, reinforcement learning, energy efficiency, system resiliency.

I. INTRODUCTION

The rapid growth of computationally intensive applications and the proliferation of mobile devices have created significant challenges for traditional cloud computing architectures. Mobile edge computing (MEC) has emerged as a promising paradigm to bring computation and storage resources closer to end-users, thereby reducing latency and improving service efficiency. By leveraging MEC, computational tasks can be offloaded from user equipment (UE) to edge servers, which enhances processing capabilities and conserves the limited battery life of mobile devices.

The increasing adoption of artificial intelligence (AI), the Internet of Things (IoT), and real-time data-driven applications further exacerbate the need for efficient edge computing solutions. Smart cities, autonomous vehicles, augmented reality (AR), and industrial automation generate vast amounts of data that require real-time processing. Traditional cloud computing infrastructures suffer from high latency and network congestion when handling these applications, making MEC an essential enabler for next-generation wireless networks.

Unmanned aerial vehicles (UAVs) have recently gained attention as mobile edge nodes that can complement terrestrial base stations (BSs) by providing dynamic coverage and computational resources [1]. UAV-assisted MEC networks enable flexible and scalable task offloading strategies, allowing UEs to offload tasks to nearby UAVs, which in turn relay them to MEC-enabled BSs. These networks have a wide range of applications, including disaster response, smart agriculture, surveillance, intelligent transportation systems, and remote healthcare. In disaster response scenarios, UAVs provide real-time data processing and communication support in areas where infrastructure is damaged or

unavailable [2]. In smart agriculture, UAVs assist in crop monitoring and precision farming by processing data on-site [3]. For surveillance and security, UAVs enhance situational awareness by analyzing video streams in real-time [4]. In intelligent transportation, UAV-assisted MEC networks optimize traffic monitoring and vehicular communication [5]. Furthermore, in remote healthcare, UAVs equipped with edge computing capabilities can facilitate medical imaging and telemedicine services by processing patient data on-site, enabling timely and resource-efficient diagnosis and decision-making [6].

Despite their advantages, UAV-assisted MEC systems face several challenges, including UAV mobility constraints, limited energy resources, dynamic wireless channel conditions, and task migration complexities. However, unlike static BSs, UAVs can dynamically adjust their positions to ensure optimal task allocation. In a number of scenarios, one of the most critical challenges is minimizing the energy consumption of UE, which directly affects device longevity and system sustainability. Efficient offloading decisions must account for the limited battery capacity of UEs, while also ensuring reliable task execution. Managing computational resources across a swarm of UAVs and BSs in a way that balances UE energy efficiency and network resiliency remains a fundamental research problem. Additionally, task redundancy and reliability become essential considerations, particularly in mission-critical applications where failures in communication or computing nodes can disrupt services.

To address these challenges, this study proposes an RL-based task offloading framework that optimizes the energy consumption of the UE while ensuring system resiliency. Since UEs typically have limited battery capacities and are more energy-constrained than infrastructure elements like UAVs or BSs, our focus is specifically on minimizing the energy consumption of the UEs rather than the UAVs themselves, which can be more easily recharged. This is particularly critical as excessive energy consumption at the UE level leads to faster battery depletion, negatively impacting user experience and service continuity.

The proposed framework introduces a redundancy mechanism based on an $(h+1)$-server permutation strategy, where tasks are redundantly assigned to multiple edge servers to guarantee execution continuity in the event of server failures. By leveraging RL, the system dynamically adapts to network conditions, balancing the trade-off between energy efficiency and failure resilience. Unlike conventional optimization techniques, RL-based approaches allow continuous learning and adaptation to dynamic environments, making them well-suited for UAV-assisted MEC networks.

While several studies have explored task offloading in UAV-assisted MEC networks, existing approaches have either focused on optimizing task distribution among UAVs and BSs, improving UAV energy efficiency, or ensuring network resiliency through redundancy mechanisms. However, to the best of our knowledge, no prior work has specifically targeted simultaneous optimization of UE energy consumption and resiliency through an $(h+1)$-server permutation strategy. This novel framework provides a unique solution by integrating RL to dynamically balance UE energy consumption and task execution reliability.

The key contributions of this work include:

- A novel UAV-assisted MEC framework enhancing UE energy efficiency by optimizing task allocation using an RL approach.
- An $(h+1)$-server permutation redundancy mechanism to improve system resiliency and mitigate the impact of task execution failures.
- A RL-based decision-making model that dynamically adapts to network conditions to optimize task offloading strategies.
- A discussion of real-world deployment challenges and potential solutions for UAV-assisted MEC networks.
- A comprehensive performance evaluation demonstrating significant improvements in energy consumption and system resiliency over a diverse set of state-of-the-art learning-based and optimization-driven MEC offloading strategies, validating the superiority and adaptability of the proposed framework under varying network conditions.

The remainder of this paper is organized as follows. Section II reviews the related work on task offloading in MEC and UAV-assisted networks. Section III describes the system model. Section IV introduces the problem formulation. Section V presents the proposed RL-based task offloading framework. Section VI presents the simulation parameters and scenarios used to evaluate the performance of the proposed approach. Section VII discusses the simulation results. Finally, Section VIII concludes the paper.

II. RELATED WORK

The existing research can be broadly classified into three main categories. The first one focuses on task offloading and resiliency in MEC networks, where studies have explored optimization techniques such as multi-objective optimization, heuristic algorithms, and machine learning to improve task allocation, minimize latency, and maximize service reliability. The second category addresses UAV-assisted MEC and fault-tolerant offloading, investigating the role of UAVs as mobile edge nodes to extend coverage and enhance task offloading efficiency. Finally, the third category focuses on energy-efficient task offloading strategies in UAV-assisted MEC systems, primarily aiming at optimizing energy consumption at different network entities, such as UAVs or UEs.

Recent works in this area have leveraged techniques like RL, distributed optimization, and cooperative UAV networks to enhance the robustness of task execution in dynamic environments. In the following, we review key contributions from all these categories, analyzing their methodologies and limitations in the context of resiliency and redundancy in MEC systems. Table 1 summarizes the main differences between

TABLE 1. Comparison of Proposed RTO Algorithm With Existing Work

Paper	Objectives	Methodology	Redundancy & Resiliency Support
Liu et al. [10]	Tradeoff between latency and reliability, task offloading efficiency	Heuristic algorithm, non-convex problem solution	✗
Chen et al. [11]	Task delivery reliability, minimize total delay	Minimum granularity decomposition, Branch and Bound algorithm	✗
Hou et al. [13]	Minimize latency, maximize reliability	Hybrid binary particle swarm optimization, expected latency definition	✗
Hu et al. [14]	Build UAV network with SDN, implement blockchain technology	Distributed control plane, blockchain for flexibility and survivability	✓ (blockchain-based survivability)
Liu et al. [7]	Maximize reliability, minimize bandwidth usage	Multi-objective optimization, RETO and DETO algorithms	✗
Malik et al. [12]	Minimize task failures, minimize energy consumption	Two-phase offloading algorithm, deferred acceptance matching	✓ (VRU fallback matching)
Peng et al. [15]	Task reliability, enhance energy efficiency	Reliability-aware offloading with shadowing for fault tolerance	✓ (shadow backup scheme)
Wang et al. [8]	Resiliency to edge server failures, task offloading efficiency	Online primal-dual algorithm, adaptive task migration	✓ (server availability aware)
Yu et al. [16]	Task load balancing, minimize network latency	Iterative algorithm, generalized Benders decomposition	✗
Zhang et al. [9]	Minimize delay, maximize service reliability	Logistic reliability model, alternative optimization method	✗
Zhao et al. [17]	Minimize energy consumption, ensure reliability	Deep Q-learning based DNN partitioning	✗
Proposed RTO Algorithm	Minimize UE energy, ensure resiliency via $(h+1)$ permutations	Q-learning with policy iteration, $(h+1)$-server permutation for task redundancy and failure recovery	✓ (full resiliency via structured $(h+1)$-server execution)

the related works and our study in terms of objectives and methodology.

A. TASK OFFLOADING AND RESILIENCY IN MEC NETWORKS

Recent advances in the MEC field have led to significant interest in resilient and efficient task offloading strategies, particularly in dynamic and heterogeneous environments. One line of research has explored how to maximize reliability during offloading while minimizing resource consumption. For instance, the work in [7] investigates a multi-edge cloud scenario designed to serve IoT applications. The authors formulate a multi-objective optimization problem to simultaneously minimize bandwidth consumption and maximize reliability, which they convert into a single-objective problem and solve using two near-optimal algorithms (named RETO and DETO).

In parallel, other studies have focused on the dynamic nature of edge resources and their impact on resiliency. In [8], a dynamic edge computing framework is presented, where end devices not only create tasks but can also function as edge servers. The edge server set is dynamic due to device mobility. To ensure reliability amid server failures, the authors introduce adaptive task offloading mechanisms, utilizing both offline batch scheduling and online primal-dual algorithms that adjust based on server availability.

Service delay and reliability have also been addressed through probabilistic modeling. In [9], the authors present a logistic-based service reliability probability model for virtual machine-based edge servers. By jointly optimizing computation resources and service quality ratios, they propose a low-complexity heuristic algorithm to solve the average reliability maximization problem efficiently. To address the trade-off between latency and reliability in distributed offloading, [10] explores multi-edge server connectivity, where each UE can offload its task to multiple edge nodes. A heuristic algorithm is proposed to make near-optimal offloading decisions that manage latency while enhancing task delivery success rates.

Focusing on ultra-reliable low-latency communications (URLLC), the authors in [11] develop a task offloading framework that ensures high reliability through a joint optimization model. They reformulate the problem as a mixed-integer program and use a branch and bound (B&B) algorithm to obtain

globally optimal solutions. Additionally, task reliability has been enhanced using virtualization of computing resources. The work in [12] introduces a two-phase offloading algorithm for IoT-to-fog computing. The authors design virtual resource units (VRUs) with variable sizes and apply a modified deferred acceptance algorithm for stable task-resource matching, followed by redistribution of unmatched resources to previously matched tasks.

A recent contribution in this domain is the work of [13], which introduces a decentralized deep RL (DRL)-based framework for task offloading and resource allocation in MEC networks. The authors consider a user-centric architecture with limited central coordination, where UE makes local decisions to minimize task execution delay and energy consumption under resource constraints. By formulating the problem as a Markov Decision Process and training a multi-agent DRL model with Proximal Policy Optimization (PPO), they demonstrate improved scalability and adaptability in dynamic environments. The proposed method also accommodates wireless channel variability and multi-user interference, yielding superior performance over centralized baselines, particularly in scenarios with high mobility and device heterogeneity.

Together, these studies provide valuable insight into task offloading under reliability constraints. However, most of them either assume static network infrastructures or overlook the constraints and energy limitations of UEs. Moreover, they do not address the combination of energy efficiency and resiliency within an RL framework—an aspect central to our proposed methodology.

B. UAV-ASSISTED MEC AND FAULT-TOLERANT OFFLOADING

Several studies have explored UAV-assisted MEC systems, focusing on fault-tolerant task offloading and resiliency. The work in [14] introduces a multi-UAV framework that ensures latency and task delivery reliability through joint optimization of communication, computation, and caching. Similarly, [15] proposes an SDN- and blockchain-based UAV network to enhance security and survivability in dynamic environments.

Other studies, such as [16], address reliability-aware computation offloading for delay-sensitive applications, utilizing a two-phase approach combining energy-efficient strategies and shadowing schemes, where backup (shadow) copies of tasks are assigned to secondary servers to ensure fault tolerance and minimize the impact of primary server failures. Furthermore, [17] employs deep Q-learning for collaborative inference among UAVs, optimizing energy consumption under reliability constraints.

In [18], the authors present a UAV-aided network leveraging mmWave backhaul to balance task load on edge servers while minimizing network latency. Their approach integrates trajectory optimization and resource allocation through iterative algorithms. Additional research includes [19], which provides a systematic mapping of UAV-assisted MEC systems, analyzing key offloading strategies and challenges. The study in [20] proposes a deep RL approach to optimize task offloading while minimizing the age of information in UAV-assisted MEC.

In [21], deep RL techniques are examined for their utility in managing computation offloading and resource allocation within UAV-assisted MEC environments, with a particular focus on the gains achieved in system efficiency and reduced energy usage. Similarly, [22] presents a systematic review on computational offloading in UAV swarm networks, focusing on local and global path planning. In [23], the authors investigate resource allocation and 3D deployment in UAV-assisted MEC networks with air-ground cooperation, proposing optimization techniques to enhance reliability and performance. Additionally, [24] discusses secure task offloading using covert communication techniques, mitigating potential threats in UAV-assisted MEC networks. In [25], the authors present a dense MARL framework for UAV-assisted vehicular networks, employing a dual-layer decision model with critical state detection to enhance convergence and coverage efficiency. Their decentralized approach improves scalability and responsiveness in dynamic environments with limited communication overhead.

C. ENERGY EFFICIENCY IN TASK OFFLOADING

Energy-efficient task offloading in UAV-assisted MEC networks has been extensively studied, with most contributions focusing on either minimizing UAV energy consumption or UE energy consumption. The majority of works target UAV energy consumption optimization. For instance, [26] proposes an RL-based offloading strategy that dynamically allocates tasks based on energy constraints and network conditions. In addition, [27] introduces a multi-agent RL approach that ensures fairness in computational task distribution while improving overall UAV energy efficiency.

Beyond UAV energy optimization, some studies focus on minimizing UE energy consumption. The work in [28] provides a comprehensive review of energy-aware task partitioning, UAV trajectory optimization, and power-efficient resource allocation. Additionally, [29] explores specific UAV-MEC systems, where reconfigurable intelligent surfaces (RIS) are leveraged to optimize signal reflection and improve energy-aware offloading performance. Moreover, [30] proposes a covert communication-based offloading strategy, ensuring secure and energy-efficient execution of tasks in UAV-assisted MEC systems through deep RL.

Further studies explore advanced resource allocation and energy harvesting mechanisms. For example, [31] presents a joint optimization framework for UAV-assisted MEC that balances energy consumption and low-latency task execution. Finally, [32] investigates wireless-powered transmission (WPT)-MEC networks, where energy harvesting techniques are integrated to extend the operational lifetime of UAV-assisted MEC systems.

While these works effectively reduce UAV and UE energy consumption, none explicitly consider resiliency in task offloading. Addressing this research gap, our study introduces an RL-based resilient task offloading strategy, integrating

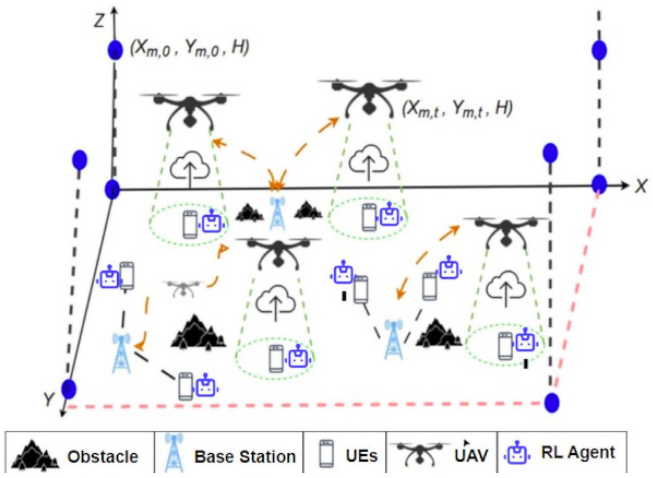

FIGURE 1. Overview of the UAV-based MEC System Model.

an $(h+1)$-server permutation model to ensure execution continuity under network failures. This dual focus on redundancy and energy efficiency differentiates our approach from existing solutions. Detailed evaluations demonstrate its effectiveness in balancing UE power consumption, resiliency, and execution reliability, making it a promising advancement in UAV-assisted MEC networks.

III. SYSTEM MODEL

As illustrated in Fig. 1, our framework considers a service area populated by a set of UEs. These UEs are served by multiple BSs, each MEC server. A fleet of UAVs further assists by offloading tasks from the UEs to the BS-based servers. Each UE is equipped with a dedicated RL agent that decides whether to process tasks locally or offload them to a nearby UAV or BS. The key notations used in our model are listed in Table 2.

The system model is based on a set of computational tasks, each with specific resource demands. These tasks originate from user devices, which can offload them to a limited subset of nearby MEC servers, each characterized by its own computational capacity. The entire system operates in discrete time slots. In each time slot t, UAVs are assumed to follow a circular mobility pattern at a fixed altitude. Specifically, each UAV moves along a predefined circular trajectory centered within the service area, ensuring that they periodically revisit the same locations. This flight mode allows UAVs to serve nearby UEs within their coverage radius as they move. The circular motion model was selected to strike a balance between persistent availability and reduced energy costs associated with continuous repositioning, reflecting a realistic but computationally manageable model for UAV-assisted offloading as proposed in [33], [34].

A. COMMUNICATION MODEL

The communication model in our system is designed to efficiently manage task offloading in a MEC environment, where multiple UEs interact with BSs and UAVs for resource allocation and task execution. The communication between these entities is controlled by RL agents, one dedicated to each UE, which make decisions based on real-time information regarding the MEC environment.

TABLE 2. Notations Table

Variable	Description
A	Height of base station antenna
$\alpha_{n,t}$	Angle of UAV n at time slot t
b_j^m	Binary variable for offloading solution C_m
C	Set of $(h+1)$-server permutations
c_s	Residual capacity of MEC server s
$c_{n,t}$	Relative UE-load of UAV n at time slot t
C_d	Switched capacitance of UE d
d	Individual UE device
D	Set of all UE devices
D_{active}	Set of active UEs
D_n^t	Set of UEs that offload tasks to UAV n at time slot t
$dist_n^t$	Distance covered by UAV n at time slot t
E_d	Total energy consumption for UE d
E_d^t	Energy level of UE d at time slot t
E_d^{local}	Total UE energy used for locally executed tasks
$E_d^{offload}$	Total UE energy used for offloaded tasks
E_d^{rx}	Sum of UE reception energy
E_d^{tx}	Sum of UE transmission energy
E_j^d	UE energy for local execution per task
$E_j^{receive}$	UE reception energy / Energy for task result reception
$E_j^{transmit}$	UE transmission energy
E_{min}, E_{max}	Minimum and maximum energy thresholds
H	Constant representing UAV height
J	Set of indivisible tasks
J_d	Set of tasks for device d
k_d^j, f_d	CPU cycles and processing frequency of UE d
n	Individual UAV
N	Set of all UAVs
N_0	Noise power spectrum density
P_m	Product of failure probabilities
$P_d^{receive}$	UE receive power
$P_d^{transmit}$	UE transmit power
$P_n^{transmit}$	UAV transmit power
$P_s^{transmit}$	BS transmit power
pr	Maximum failure probability
p_o	Probability of server failure
r_j	Resource demand of task j
R_j	Task result size
r_{dn}	Data rate between UE and UAV
r_{ds}	Data rate between UE and BS
r_{nd}	Data rate between UAV and UE
r_{ns}	Data rate between UAV and BS
r_{sd}	Data rate between BS and UE
r_{sn}	Data rate between BS and UAV
s	Individual base station (BS)
S	Set of all BSs
S_d	Set of accessible servers for device d
T	Set of time instances
t_j^{return}	Task return time
t_j^{route}	Task routing time
$t_j^{route(dn)}$	Transmission time from the UE to the UAV
$t_j^{route(ds)}$	Transmission time from the UE to the BS
$t_j^{route(ns)}$	Transmission time from the UAV to the BS
W^u	Minimum distance between UAVs
W^{max}	Maximum UAV coverage radius
$W_{d,n,t}$	Distance between UE d and UAV n at time slot t
$W_{n,n',t}$	Distance between UAV n and n' at time slot t
$W_{n,s,t}$	Distance between UAV n and BS s at time slot t
X_d, Y_d	Coordinates of UE d
$X_{n,t}, Y_{n,t}$	Coordinates of UAV n at time slot t
X_s, Y_s	Coordinates of BS s
z_d^t	Offloading decision for UE d at time slot t
β_0	Channel power gain at 1m reference distance

1) USER EQUIPMENTS (UES)

Each UE is considered to be a mobile device with computational tasks that may exceed its processing capability or energy constraints. These UEs are equipped with a dedicated RL agent that continuously monitors the UE energy level and the total number of tasks being processed on all BSs. Based on these parameters, the RL agent determines whether to offload the task directly to a MEC server, use an offloading UAV, or to process it locally.

2) BASE STATIONS (BS)

BS are stationary nodes that provide computational resources for UEs through co-localized MEC servers. They receive tasks from UEs directly or via UAVs. Each BS has a computation capacity and a number of tasks being processed on it. Communication between UEs and BS, or between UAVs and BS, occurs over a dedicated wireless link, and the BS are assumed to have sufficient power to handle multiple offloaded tasks simultaneously.

3) UNMANNED AERIAL VEHICLES (UAVS)

UAVs serve as intermediary nodes that assist in task offloading, especially in scenarios where direct communication between UEs and BS is problematic due to distance or signal propagation issues. UAVs can temporarily store and relay computational tasks from UEs to BS. UAVs provide flexibility and increase the coverage area for task offloading, ensuring that UEs in remote or dynamic radio environments can still offload tasks effectively.

4) DECISION-MAKING PROCESS

The RL agents utilize a reward-based mechanism to optimize the task offloading process. Rewards are calculated based on the UE energy efficiency. The goal of each RL agent is to maximize cumulative rewards over time, ensuring that the UE battery life is prolonged while maintaining high computational performance. This communication model is designed to be adaptive, scalable, and resilient, providing an effective solution for task offloading in dynamic and heterogeneous MEC environments.

B. PLACEMENT MODEL

The distances between the entities in the system are defined as follows:

1) Considering static UEs, the distance separating UE d and UAV n in time slot t:

$$W_{d,n,t} = \sqrt{(X_{n,t} - X_d)^2 + (Y_{n,t} - Y_d)^2 + H^2} \quad \forall\, d \in D,\ n \in N,\ t \in T \tag{1}$$

2) At time slot t, the distance between UAV n and BS s:

$$W_{n,s,t} = \sqrt{(X_{n,t} - X_s)^2 + (Y_{n,t} - Y_s)^2 + H^2} \quad \forall\, n \in N,\ s \in S,\ t \in T \tag{2}$$

3) At time slot t, the distance between UAVs n and n' :

$$W_{n,n',t} = \sqrt{(X_{n,t} - X_{n',t})^2 + (Y_{n,t} - Y_{n',t})^2} \quad \forall\, n, n' \in N, t \in T \tag{3}$$

4) Distance between UE d and BS s:

$$W_{d,s} = \sqrt{((\|X_d - X_s\|)^2 + (\|Y_d - Y_s\|)^2)} \quad \forall\, d \in D,\ s \in S \tag{4}$$

C. DELAY CONSIDERATIONS

The communication time for offloading a task j is given as:

1) When routed through an UAV n to reach the BS s:

$$\begin{aligned} t_j^{route} &= t_j^{route(dn)} + t_j^{route(ns)}, \\ t_j^{route(dn)} &= \frac{I_j}{r_{dn}}, \\ r_{dn} &= B \cdot \log_2\left(1 + \frac{P_d^{transmit} \cdot \beta_0^d}{B \cdot N_0 \cdot W_{d,n,t}^2}\right), \\ t_j^{route(ns)} &= \frac{I_j}{r_{ns}}, \\ r_{ns} &= B \cdot \log_2\left(1 + \frac{P_n^{transmit} \cdot \beta_0^n}{B \cdot N_0 \cdot W_{n,s,t}^2}\right) \end{aligned} \tag{5}$$

2) When directly offloaded to the BS:

$$\begin{aligned} t_j^{route} &= t_j^{route(ds)}, \\ t_j^{route(ds)} &= \frac{I_j}{r_{ds}}, \\ r_{ds} &= B \cdot \log_2\left(1 + \frac{P_d^{transmit} \cdot \beta_0^d}{B \cdot N_0 \cdot W_{d,s,t}^2}\right), \end{aligned} \tag{6}$$

3) Once a task is executed, if the result is returned via the UAV, the result return delay is:

$$\begin{aligned} t_j^{return} &= t_j^{return(sn)} + t_j^{return(nd)}, \\ t_j^{return(sn)} &= \frac{R_j}{r_{sn}}, \\ r_{sn} &= B \cdot \log_2\left(1 + \frac{P_s^{transmit} \cdot \beta_0^s}{B \cdot N_0 \cdot W_{n,s,t}^2}\right), \\ t_j^{return(nd)} &= \frac{R_j}{r_{nd}}, \\ r_{nd} &= B \cdot \log_2\left(1 + \frac{P_n^{transmit} \cdot \beta_0^n}{B \cdot N_0 \cdot W_{d,n,t}^2}\right) \end{aligned} \tag{7}$$

4) If the result is returned via the BS directly to the UE, the result return delay is:

$$t_j^{return} = t_j^{return(sd)},$$
$$t_j^{return(sd)} = \frac{R_j}{r_{sd}},$$
$$r_{sd} = B \cdot \log_2\left(1 + \frac{P_s^{transmit} \cdot \beta_0^s}{B \cdot N_0 \cdot W_{d,s}^2}\right) \quad (8)$$

5) The end-to-end (E2E) delay is:

$$T_j^{E2E} = t_j^{route} + t_j^{return} \quad (9)$$

D. ENERGY CONSIDERATIONS

The total energy consumed by a UE d is:

$$E_d = E_d^{offload} + E_d^{local}, \quad (10)$$

where,

$$E_d^{offload} = E_d^{tx} + E_d^{rx},$$
$$E_d^{tx} = \sum_{s\in S}\sum_{j\in J_d}\sum_{t\in T} E_j^{transmit},$$
$$E_d^{rx} = \sum_{s\in S}\sum_{j\in J_d}\sum_{t\in T} E_j^{receive},$$
$$E_d^{local} = \sum_{j\in J_d}\sum_{t\in T} E_j^{d},$$
$$E_j^{d} = C_d \cdot k_d^j \cdot f_d^2,$$
$$E_j^{transmit} = P_d^{transmit} \cdot (t_j^{route(dn)} + t_j^{route(ds)}),$$
$$E_j^{receive} = P_d^{receive} \cdot (t_j^{return(nd)} + t_j^{return(sd)}) \quad (11)$$

E. DECISION VARIABLES

We are using a binary decision variable for offloading :

$$z_d^t = \begin{cases} 1, & \text{if task is offloaded} \\ 0, & \text{otherwise.} \end{cases} \quad (12)$$

For this variable, $z_d^t = 1$ when UE d chooses to offload the task in time slot t and $z_d^t = 0$ when UE d decides to compute the task locally.

This allows us to define the relative UE-load of an UAV at time t:

$$c_{n,t} = \frac{\sum_{d\in D_n^t} z_d^t}{|D_n^t|}, \quad \forall t \in T, \quad n \in N \quad (13)$$

where D_n^t reflects the dynamic association due to UAV mobility.

IV. PROBLEM FORMULATION

Our main problem can be described as an ***Edge Server Redundancy and Resiliency problem***, that aims at maximizing the UE task throughput on MEC servers by providing redundant and resilient solutions for task offloading while minimizing the UE energy consumption.

A. EDGE SERVER REDUNDANCY AND RESILIENCY

System throughput is quantified by aggregating the computational demands, such as CPU cycles, of tasks promptly executed on edge servers after successful assignment. A MEC system is more prone to errors and failures than a classical cloud. At the same time, there are many scenarios where task execution can not cope with such computation failures. Therefore, we introduce redundancy in the task-offloading process by assigning a task j to multiple BS servers simultaneously. If some of the servers fail, this mechanism ensures the task is still executed by the functional ones. Task offloading remains, of course, constrained by the capacity of the edge servers; the total workload assigned to a server must not exceed its available resources. To further enhance system resilience, we introduce an additional constraint that ensures each assigned task is guaranteed to execute successfully, even if up to h edge servers experience failures (e.g., due to mobility or power loss). Here, h represents a configurable parameter that dictates the system's tolerance to server failures.

In the event that h edge servers simultaneously fail, the tasks originally assigned to them must be reassigned to the remaining operational servers. It might initially seem sufficient to simply ensure that the cumulative residual capacity of the $(m - h)$ operational servers exceeds the total demand of the tasks from the failed servers. However, this overlooks the constraint that tasks are indivisible; thus, it is not only essential that the total available capacity is adequate, but also that each individual task can be accommodated by the servers still active in the system. Given the dynamic nature of edge computing environments, it is crucial to guarantee the successful execution of all tasks, even under the failure of up to h edge servers. Let us consider the example where $h = 1$. Suppose an arbitrary task j is initially offloaded to an edge server $s_0 \in S$. For task j to continue executing seamlessly even if a server fails, it is essential to have sufficient residual resource capacity. This means that the task must be assigned to an additional edge server, running in parallel with the original one.

Therefore, to achieve **redundancy**, our robust task offloading strategy assigns a pair of edge servers, s_m and $s_{m'}$, to ensure that task j remains executable even if one server fails. Both servers execute the task simultaneously, allowing for seamless failure recovery. To achieve **resiliency**, the system extends this redundancy mechanism by ensuring that if up to h servers fail, there remains at least one operational backup server to execute the task. Specifically, if all prior h assigned servers fail, the system must ensure that sufficient resources are available on a backup server.

Thus, the offloading solution for task j involves assigning it to $(h + 1)$ edge servers, forming what we refer to as an $(h + 1)$-server permutation. This guarantees execution resilience despite multiple simultaneous failures, provided that $h + 1 \leq |S|$, where $|S|$ represents the total number of available

edge servers. By incorporating both redundancy (simultaneous execution on two servers) and resiliency (fallback to an $(h+1)$-server permutation), this approach ensures robust task execution even in highly dynamic and failure-prone edge computing environments.

To simplify the notation, let $C = \{C_1, C_2, \ldots, C_{|C|}\}$ represent the complete set of all possible $(h+1)$-server permutations. Each individual permutation within this set is denoted as $C_m \subset C$. By way of illustration, if $h = 1$, tasks are initially allocated to a permutation consisting of $(h+1) = 2$ servers. This means that the task is simultaneously executed on two servers, denoted as:

$$C_m = \{s_0, s_1\}$$

where s_0 and s_1 are both actively running the task to ensure redundancy. In the event that one of these two servers fails, the task execution continues uninterrupted on the remaining operational server. However in this case, if both s_0 and s_1 fail simultaneously, the system escalates the redundancy mechanism by reassigning the task to a new set of backup servers, forming an (h+2)-server permutation:

$$C'_m = \{s_2, s_3, s_4\}$$

This reassignment guarantees that even in the worst-case scenario where both primary servers fail, at least one of the newly allocated backup servers will remain operational, thereby ensuring task resiliency.

In another example, when $h = 2$, the initial task assignment follows an $(h+1)$-server permutation, meaning the task is distributed across three servers:

$$C_m = \{s_0, s_1, s_2\}$$

In this scenario, s_0 and s_1 execute the task simultaneously for redundancy, while s_2 is designated as a backup server, ready to take over execution if either of the two primary servers fails. The presence of this additional backup enhances fault tolerance by ensuring that the system remains functional even if one of the actively executing servers fails. However, if all three servers $\{s_0, s_1, s_2\}$ fail simultaneously, the system further escalates its resiliency mechanism by dynamically reassigning the task to a new (h+2)-server permutation:

$$C'_m = \{s_3, s_4, s_5, s_6\}$$

This reassignment ensures that the task remains executable even in the presence of multiple server failures, significantly improving system reliability in highly dynamic and failure-prone edge computing environments.

The binary variable $b_j^m \in \{0, 1\}$ indicates whether the $(h+1)$-server permutation C_m^j is the offloading solution for j ($b_j^m = 1$) or not:

$$\mathcal{C}1 : b_j^m \in \{0, 1\}, \quad \forall J, C_m \tag{14}$$

Since all tasks are indivisible, each task j can only be assigned to a single $(h+1)$-server permutation C_m, as shown :

$$\mathcal{C}2 : \sum_{C_m \in C} b_j^m \leq 1, \quad \forall j \in J \tag{15}$$

Moreover, we introduce p_s as the probability of failure of a server s, where $s \in C_m$, and P_m is the probability of failure of all servers in C_m, i.e. $P_m = \pi_{s \in C_m} p_s$. To guarantee that the task is successfully assigned to MEC servers that are not expected to go out of service, P_m should be bounded by pr. The failure probability constraint is expressed as:

$$\mathcal{C}3 : P_m \cdot b_j^m \leq pr, \ \forall j \in J \tag{16}$$

where pr is a user-defined value.

To indicate the feasibility of using a particular $(h+1)$-server permutation, we introduce a binary variable l_m. This variable is set to 0 if there exists an edge server s within the permutation C_m such that $s \in C_m$ and $p_s \geq \epsilon$, indicating that all servers in the permutation meet the required operational threshold. Conversely, l_m is set to 1 if this condition is not met. Consequently, if $l_m = 0$, the permutation C_m is deemed unsuitable for task offloading, as at least one of its constituent edge servers does not satisfy the specified failure probability constraint for the end device generating task j.

Since task j is executed simultaneously on two primary servers in C_m under normal conditions (i.e., without failure), we define the initial load distribution among the primary servers. The task execution load on the first assigned server s is given by:

$$\sum_{C_m} b_j^m \cdot r_j = \lambda_s \cdot c_s \tag{17}$$

where $\lambda_s (\leq 1)$ represents the load ratio of server s, and c_s denotes its computational capacity. To maintain system equilibrium and adhere to resource capacity limitations, the load ratio on each edge server must be kept at or below 1. The subsequent constraint ensures this condition is met:

$$\mathcal{C}4 : \sum_{j \in J} \sum_{C_m \in C} b_j^m \cdot r_j \leq c_s, \quad \forall s \in S \tag{18}$$

B. MINIMIZATION PROBLEM

After discussing the above problem, we can now formulate the main minimization problem as follows:

$$\min_{\{z_d^t\}} \sum_{t \in T} \sum_{d \in D} z_d^t \cdot E_d$$
$$\text{s.t.} \quad \mathcal{C}1\text{–}\mathcal{C}4 \tag{19}$$

where:

- $\sum_{d \in D} z_d^t \cdot E_d$ accounts for the total energy consumption across all UEs participating in the offloading process.
- The objective function aims to minimize the UE energy consumption by optimizing the allocation of tasks.
- Constraints $\mathcal{C}_1$ to $\mathcal{C}_4$ enforce system-wide limits on capacity, task allocation, and failure recovery mechanisms.

By solving this optimization problem, the system ensures that:

1) Tasks are executed with redundancy on two primary servers.
2) In case of failure, tasks are reassigned to a resilient server permutation.
3) UE energy consumption is minimized while maintaining feasible computational constraints.

V. RL TASK OFFLOADING (RTO) ALGORITHM

While the use of reinforcement learning for resource allocation is established, the novelty of our RTO algorithm lies in three key areas. First, we design an RL agent that operates within the formal constraints of our $(h+1)$-server permutation strategy, a unique integration of learning and structured resiliency. Second, our reward function is specifically engineered to enforce this resiliency by heavily penalizing decisions that fail to secure redundant servers. Finally, our framework is implemented as a fully decentralized multi-agent system, enhancing scalability and responsiveness compared to centralized approaches.

The proposed RTO algorithm employs an RL approach to optimize task offloading in a MEC network. This network comprises UE, BS, and UAVs, each playing a crucial role in minimizing UE energy consumption and optimizing task distribution by providing UE with access to a pair of BS for processing the offloaded task simultaneously (achieving redundancy) and providing an $(h+1)$ permutation set in case of failure of both servers (achieving resiliency). The environment models the dynamic interactions among these elements, providing a realistic setting for the RL agent to learn and make decisions.

A. RL AGENT DEFINITION

In this work, each UE is equipped with a dedicated RL agent. The primary responsibility of these agents is to make decisions regarding task offloading to minimize UE energy consumption while ensuring efficient task execution. The RL agent interacts with the environment through the network state to retrieve knowledge about the total number of tasks being processed on BSs, as well as the energy level of the corresponding UE. The RL agent aims to learn an optimal policy through which it can maximize long-term rewards by consistently selecting the most effective offloading strategy at each decision epoch.

The proposed RTO framework can also be interpreted from a Multi-Agent RL (MARL) perspective, where each UE operates as an independent learning agent within a shared MEC-UAV environment. These agents interact indirectly through the edge network and make decentralized offloading decisions based solely on their local observations, such as the UE's residual energy. This design adheres to typical MARL paradigms, where multiple agents learn and act simultaneously without requiring centralized coordination.

Unlike centralized MARL approaches that rely on global state awareness or parameter sharing, our framework adopts a fully decentralized scheme. Each agent learns its policy independently, which facilitates scalability and adaptability as the number of UEs increases.

By embedding learning agents at the UE level, our RTO framework combines low coordination overhead with high responsiveness to real-time variations such as network congestion or task failures. This enables the system to maintain resilience while continuously optimizing energy efficiency and offloading performance across heterogeneous and dynamic network topologies.

The environment is modeled as a Markov Decision Process (MDP), where the state represents the current conditions of the network, the actions correspond to offloading decisions, and the rewards provide feedback on the effectiveness of those actions. Each RL agent observes the state of its associated UE and the available network resources, before deciding whether to offload a task or process it locally. The agent updates its knowledge based on the rewards received from its actions, continually improving its policy to achieve better performance over time. The interaction between the RL agent and the environment can be summarized as follows: at each time step, the agent observes the state, selects an action based on its policy, receives a reward, and transitions to a new state. This process is repeated for multiple episodes, allowing the agent to explore different strategies and learn from their experience. By the end of the training phase, the RL agent should have learned a policy that effectively balances the trade-offs between energy consumption and task completion efficiency.

B. STATE SPACE

In RL, the **state** represents the current condition of the environment as perceived by the agent. The state space $STATE$ consists of all possible states that the system can experience. At each time step t, the state of agent d is denoted as $state_{d,t}$, which includes the UE's remaining energy $E^{(t)}_{d(remain)}$ and the total number of tasks currently being processed on all BSs TT^t_{BS}. This formulation ensures that each UE observes a personalized state while being informed about the shared network load. For our UAV-assisted MEC task offloading framework, the state $state_{d,t}$ is denoted as:

$$state_{d,t} = \{E^t_{d(remain)}, TT^t_{BS}\} \tag{20}$$

where:

- $E^t_{d(remain)}$: The remaining energy level of UE d at t.
- TT^t_{BS}: The total number of computational tasks being processed on all BSs at t.

C. ACTION SPACE

Each UE agent chooses between two actions:

$$action_{d,t} = \begin{cases} \text{Local Execution}, & \text{if } z^t_d = 0, \\ \text{Offload to UAV or BS}, & \text{if } z^t_d = 1, \end{cases} \tag{21}$$

The two actions can be described as follows:

- *Offload:* The UE attempts to offload its task to either the nearest UAV or BS. The role of the UAV is to act as an

intermediary, collecting tasks from UEs and delivering them to a BS for processing. If both a UAV and a BS are available, the UE identifies the nearest UAV or BS and chooses it to be the offloading destination.
- *Local Execution:* The UE processes the task locally, consuming its own energy.

D. REWARD FUNCTION

The reward function R is a critical component of the RL agent's decision-making process. It guides the agent towards actions that minimize UE energy consumption and achieve efficient task completion. The reward function is designed to achieve three main objectives: *i)* encourage minimum UE energy consumption during task offloading, *ii)* penalize situations where the RL agent fails to offload the task and processes it locally, and *iii)* encourage assigning a pair of BS for the UE offloading process while maintaining resiliency by preparing an $(h+1)$ permutation set for the UE in case the pair of BSs fails simultaneously. The reward function R is defined as follows:

$$R = \begin{cases} \exp\left(\frac{x}{a} - b\right) + c, & \text{if offloading succeeds,} \\ -\exp\left(\frac{x}{a} - b\right) + c, & \text{if processed locally,} \\ -\exp\left(\frac{100}{a} - b\right) + c, & \text{if offloading fails.} \end{cases} \quad (22)$$

where:
- $x = 100 - E_{remain}$ represents the energy consumed since the initial state.
- E_{remain} is the remaining energy of the UE after the current task is processed.
- a, b, c are tunable parameters controlling reward scaling and sensitivity.

The reward function in the proposed algorithm is designed to balance energy consumption and task completion efficiency, ensuring a trade-off between these factors. The function assigns different reward values depending on the outcome of the task execution process. If a task is successfully offloaded, the RL agent receives a positive reward, encouraging it to continue utilizing offloading strategies when beneficial. This reward is computed using an exponential function, where the consumed energy relative to the remaining one influences the magnitude of the reward.

When an RL agent chooses to execute a task locally, the reward function applies a negative exponential term, discouraging local processing due to its higher energy cost. The level of penalization is determined by the amount of energy consumed, ensuring that local execution is used only when necessary. The lowest reward is assigned when offloading fails due to network congestion or when a pair of BSs is not available, as the algorithm mandates that each task must be assigned to two servers to guarantee execution reliability. This strict penalization guides the learning process toward selecting offloading strategies that prioritize redundancy and system resiliency while optimizing energy efficiency.

The failure resiliency mechanism within the reward function incorporates an $(h+1)$-server redundancy strategy. If the primary offloading servers fail, the task is automatically reassigned to an alternative set of servers, ensuring uninterrupted execution. This constraint reinforces system robustness by penalizing offloading strategies that increase failure risks while rewarding those that maintain execution continuity.

E. ALGORITHM DESCRIPTION

The proposed algorithm follows an RL framework that dynamically adjusts task allocation between local execution, UAV offloading, and resilient $(h+1)$-server permutations. The different steps are presented in Algorithm 1.

1) INITIALIZATION

At the start, the algorithm establishes several key input parameters, notably D, which represents the number of UEs, and N, which denotes the count of UAVs functioning as edge relay nodes, and the maximum number of server failures h. The RL parameters $(\alpha, \gamma, \epsilon)$ are also initialized, where α represents the learning rate, γ is the discount factor that determines how future rewards influence learning, and ϵ controls the balance between exploration and exploitation. The Q-tables $Q(state_{d,t}, action_{d,t})$, which store state-action values for each UE, are initialized to enable learning-based decision-making.

2) TRAINING EPISODES

The algorithm runs for E_{max} episodes, with each episode representing an independent training cycle. At the start of each episode, the environment is reset, and the system initializes to its starting state.

3) TASK EXECUTION (LINES 4-6)

For every time step within T_{max}, the algorithm gives each UE's RL agent the autonomy to assess prevailing network conditions and independently choose between local processing and task offloading.

4) UE ACTION SELECTION (LINE 7)

Each UE selects an action $action_{d,t}$ using an ϵ-greedy policy. This policy ensures that the RL agent selects the action with the highest known Q-value most of the time, while occasionally exploring alternative actions with probability ϵ. The decision is based on the current state $state = \{E^t_{d(remain)}, TT^t_{BS}\}$ (Line 6), which provides information about the remaining UE energy and the total number of tasks being processed by all BSs.

5) TASK EXECUTION AND REDUNDANCY MECHANISM (LINES 8-24)

Once the action $action_{d,t}$ is selected, the UE proceeds to execute the task accordingly. If the chosen action is to offload, the UE first identifies all available UAVs and BSs within communication range. From this set, the two closest edge servers, denoted as s_0 and s_1, are selected. The task is then offloaded to both servers simultaneously to ensure redundancy. That is,

Algorithm 1: Resilient Task Offloading (RTO) Algorithm with $(h+1)$-Server Redundancy.

Input : Set of UEs D, UAVs N, max failures h, learning parameters $(\alpha, \gamma, \epsilon)$
Output: Optimized task offloading decisions ensuring failure resilience

1 Initialize Q-tables $Q(state_{d,t}, action_{d,t})$ for all UEs;
2 **for** *episode* $e = 1$ *to* E_{max} **do**
3 Reset environment and initialize state $state$;
4 **for** *time step* $t = 1$ *to* T_{max} **do**
5 **for** *each UE* $d \in D$ **do**
6 Observe state $state_{d,t} = \{E^{(t)}_{d(remain)}, TT^{(t)}_{BS}\}$;
7 Select action $action_{d,t}$ using ϵ-greedy policy;
8 **if** *Offload* **then**
9 Identify nearest available UAV or BS;
10 **if** *UAVs or BS available* **then**
11 Assign task to two servers $\{s_0, s_1\}$;
12 **if** s_0 *or* s_1 *fails* $(h = 1)$ **then**
13 Reassign to $(h+1)$ permutation $C'_m = \{s_2, ..., s_{h+1}\}$;
14 **else**
15 **if** $h = 2$ **then**
16 Reassign to (h+2) permutation $C'_m = \{s_3, ..., s_{h+2}\}$;
17 Compute offload energy and update state $state_{d,t+1}$ using (10);
18 Compute reward using Eq. (22);
19 **else**
20 Offloading failed; execute locally;
21 Update energy and compute reward;
22 **else**
23 Execute task locally;
24 Update energy and compute reward;
25 Update Q-values using Eq. (24);
26 Store cumulative reward for the episode;

both s_0 and s_1 execute the task in parallel, providing fault tolerance in case one of the servers fails or becomes unreachable during execution.

This parallel execution ensures that the task completes successfully as long as at least one of the two servers finishes processing it. Once the first result is received, the redundant output from the second server can be discarded or ignored, depending on system policy.

If no UAVs or BSs are available at the current time step—either due to range limitations or lack of computational capacity—the UE executes the task locally. While this consumes more energy, it guarantees task completion when offloading is not feasible.

If either of these servers fails (i.e. $h = 1$), the system reassigns the task to an $(h+1)$-server permutation. The new set of servers is defined as:

$$C'_m = \{s_2, \ldots, s_{h+1}\} \tag{23}$$

Once the task is offloaded, the UE computes the offloading energy consumption using (10). At the next time step, the UE observes the updated state $state_{d,t+1} = \{E_d^{(t+1)}(remain), TT_{BS}^{t+1}\}$, which reflects the remaining energy and system load after action execution. The reward is then computed using (22) to provide feedback on the effectiveness of the decision.

If both servers $\{s_0, s_1\}$ failed (i.e. $h = 2$), the task must be reassigned to an (h+2)-server permutation, ensuring execution resilience. The system updates the state accordingly and computes the corresponding reward.

If offloading fails completely, the UE processes the task locally, updating its energy consumption using (10). The new state is recorded, and the reward function penalizes local execution due to the high energy cost.

6) Q-VALUE UPDATE AND STATE TRANSITION (LINES 25-26)

After task execution, the Q-values are updated using the Bellman equation:

$$\begin{aligned} Q_d(state_{d,t}, action_{d,t}) \leftarrow\ & Q_d(state_{d,t}, action_{d,t}) + \alpha \big[R_{d,t} \\ & + \gamma \max_{action_{d,t+1}} Q_d(state_{d,t+1}, \\ & action_{d,t+1}) - Q_d(state_{d,t} \\ & , action_{d,t})\big] \end{aligned} \tag{24}$$

The environment then transitions to the next state, incorporating the impact of the latest action on the system status.

After each episode, cumulative rewards are stored, allowing the RL agent to improve its offloading decisions over successive training cycles.

To summarize, the proposed algorithm ensures failure-resilient offloading by enforcing task allocation to at least two servers. The state-aware RL framework dynamically adjusts decisions based on UE energy levels and system load. The reward function penalizes failed offloading and local execution while reinforcing energy-efficient and resilient task offloading to MEC servers, directly or through UAVs. The framework is scalable and adaptable to any network deployment.

F. COMPUTATIONAL COMPLEXITY ANALYSIS

The computational complexity of the proposed RTO framework can be analyzed in two parts: the RL agent's internal decision-making process and the complexity of executing an action within the environment, which includes the core resiliency logic.

1) RL AGENT COMPLEXITY (DECISION-MAKING)

The RTO algorithm is designed to be lightweight at the agent level. For each of the $|D|$ agents, the core operations are computationally efficient:

- *Action Selection:* In the execution phase, an agent selects the best action by comparing the Q-values for its current state. This requires a simple lookup and comparison across the $|A|$ possible actions, resulting in a complexity of $O(|A|)$.
- *Q-Value Update:* During training, the Bellman update also requires finding the maximum Q-value for the next state, which has a complexity of $O(|A|)$.

Since the action space in our work is binary ($|A| = 2$), the agent's internal complexity is constant, $O(1)$, per step.

2) ACTION EXECUTION COMPLEXITY (RESILIENCY LOGIC)

The primary computational cost of our framework lies in executing an offload action. When an agent chooses to offload, the system must find the optimal path for the task, which can be either a direct connection to a BS-hosted server or a relayed path through a UAV.

- *Initial Path Selection:* To ensure redundancy, the strategy requires identifying the two best initial connection points. For a single UE, this involves a search across all potential first hops: the $|S|$ Base Stations (for a direct path) and the $|N|$ UAVs (for a relayed path). Finding the two closest points from this combined pool of $|N| + |S|$ entities has a linear search complexity of $O(|N| + |S|)$.
- *Failure Recovery:* In the event of a server failure, the resiliency mechanism is triggered. The algorithm must then re-search the pool of available entities to form a new $(h + 1)$ or $(h + 2)$ permutation, which incurs a similar complexity of $O(|N| + |S|)$.

3) OVERALL SYSTEM COMPLEXITY

The complexity of the RL agent itself is negligible. The dominant cost is the path selection process that must be executed for each of the $|D|$ UEs that choose to offload in a given time step. Therefore, the overall computational complexity of the RTO framework per decision cycle is:

$$O(|D| \cdot (|N| + |S|)) \tag{25}$$

This shows that the algorithm's runtime scales linearly with the number of UEs and the total number of potential connection points (UAVs and BSs). This linear scalability makes the RTO framework efficient and practical for deployment.

VI. SIMULATION PARAMETERS AND SCENARIOS

We conduct simulations using a custom-developed environment written in Python, which models the interactions between UEs, BSs, and UAVs. The following parameters and configurations were used throughout the simulations to evaluate the system performance.

TABLE 3. Simulation Parameters

Parameter	Value
Network Layout	
Simulation Area	2×2 Km grid
Number of UEs ($\|D\|$)	20 (varied in Fig. 12)
Number of BSs ($\|S\|$)	10 (varied in Fig. 13)
Number of UAVs ($\|N\|$)	5 (varied in Fig. 14)
UAV Height (H)	50 m
BS Capacity (C_s)	5 tasks
Task and UE Parameters	
Task Size (I_j)	$\mathcal{U}(0.1, 1.0)$ MB
Task Result Size (R_j)	$\mathcal{U}(0.01, 0.02)$ MB
Initial UE Energy	100 units
CPU Cycles per bit (k_d^j)	1000 cycles/bit
Switched Capacitance (C_d)	10^{-28} F
UE CPU Frequency (f_d)	1.0 GHz
Wireless Channel Parameters	
Bandwidth (B)	1 MHz
Noise Power Density (N_0)	-174 dBm/Hz
Channel Gain @ 1m (β_0)	-30 dB
UE Transmit Power ($P_d^{transmit}$)	0.5 W
UE Receive Power ($P_d^{receive}$)	0.1 W
UAV Transmit Power ($P_n^{transmit}$)	0.5 W
BS Transmit Power ($P_s^{transmit}$)	1.0 W
RL Agent Parameters	
Learning Rate (α)	0.1
Discount Factor (γ)	0.9
Exploration Rate (ϵ)	1.0
Exploration Decay	0.999999

A. SIMULATION PARAMETERS

We evaluate the performance of our offloading framework by simulating a MEC environment with UEs, BSs, and UAVs. The simulation takes place in a 2×2 km grid, where UEs generate computational tasks that may either be processed locally or offloaded to nearby UAVs or BSs. The environment is designed to capture realistic constraints, including energy consumption, bandwidth limitations, and processing capacities of the MEC infrastructure. Additionally, RL agents control the offloading decisions based on predefined parameters that influence learning behavior and convergence. The detailed simulation parameters are presented in Table 3.

The parameters in Table 3 are selected to represent a realistic operational scenario. The network layout and entity counts model a moderately dense suburban environment, and their impact is further explored in our sensitivity analysis (Section VII-E). Hardware-specific values, such as UE CPU frequency and transmit power, are set to typical values for modern mobile devices. Key wireless parameters are grounded in established literature; for instance, the 1 MHz channel bandwidth and -174 dBm/Hz noise power density are standard choices aligned with foundational UAV-MEC studies like [35]. This comprehensive and justified parameter set ensures the validity and reproducibility of our results.

B. SCENARIOS FOR COMPARISON

To evaluate the proposed system, which leverages UAVs for task offloading, we compared its performance against three alternative benchmarks: *i)* no UAVs scenario, *ii)* no local execution scenario, and *iii) No Offload* scenario. The details of each scenario are as follows:

1) PROPOSED RTO ALGORITHM SCENARIO

In this UAV-assisted scenario, $|N| > 0$ UAVs are available to assist UEs by offloading tasks. UAVs dynamically move in a predefined circular motion and deliver the tasks to the nearest two available BSs to achieve redundancy. The offloading decision is made by an RL agent attached to each UE based on the trade-off between local processing and offloading to UAVs or BSs. We aim to minimize the energy consumed by UEs while ensuring tasks are completed efficiently.

2) NO UAV SCENARIO

This baseline represents the system without UAV support, i.e., $N = 0$. In this scenario, tasks can only be offloaded to BSs or processed locally.

3) *NO LOCAL PROCESSING* SCENARIO

In this benchmark, local processing is disabled, and all tasks must be offloaded. UEs rely entirely on BSs and UAVs for task execution. This scenario evaluates the system performance when local processing is not an option, highlighting the dependency on the availability of UAVs and BSs for task completion. If a task fails due to network congestion or unavailability of computational resources, it is considered a failed task and not rescheduled for future time slots. This ensures that the evaluation reflects the ability of the system to complete tasks under constrained offloading conditions without retrying delayed tasks.

4) *NO OFFLOAD* SCENARIO

In this scenario, UEs are restricted from offloading their tasks and must process them locally. Without any offloading capability, UEs rely solely on their own computational resources to execute tasks. This scenario evaluates the system performance under conditions where offloading is not an option, emphasizing the impact of local execution constraints. If a task cannot be completed within the current time slot due to limited computational resources, it is delayed to the next time slot instead of being considered a failed task. This models real-world scenarios where tasks accumulate when processing capacity is insufficient.

The simulations compare these scenarios across multiple metrics, including average UE energy levels, cumulative rewards, and task completion efficiency. Sensitivity analysis is performed by varying key parameters, such as the number of UEs, BSs, UAVs, and task sizes, to evaluate the robustness of the proposed system under different operational conditions.

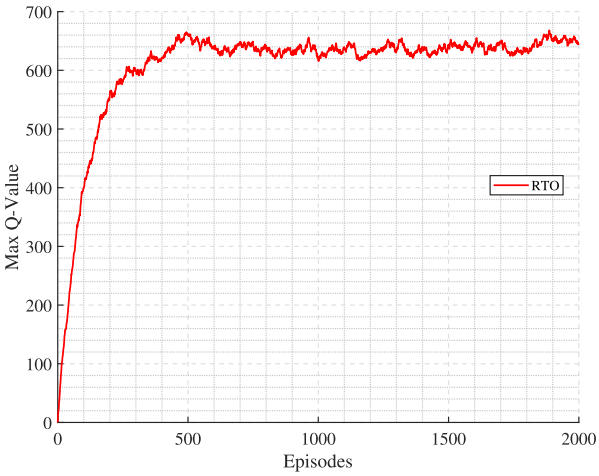

FIGURE 2. Convergence of the RTO algorithm across episodes.

The simulations span a total of 2000 episodes, with task offloading decisions made based on the learned Q-values of the RL agents. These parameters are used to analyze the impact of UAVs, offloading strategies, and various system configurations on energy consumption and task completion efficiency.

VII. SIMULATION RESULTS

This section details the outcomes of our simulation experiments, analyzed using metrics such as energy consumption, cumulative rewards, and task processing success. The discussion is organized as follows: we first demonstrate the convergence of the proposed RTO algorithm to validate its learning stability. Next, to address the role of multi-agent collaboration, we introduce a cooperative MARL baseline (C-RTO) and compare its performance against our proposed RTO framework. We then evaluate our RTO algorithm against several other baseline scenarios. Finally, we conduct a comprehensive sensitivity analysis, measuring our framework's performance against state-of-the-art benchmarks under various network conditions.

A. CONVERGENCE OF THE RTO ALGORITHM

The foundation of our proposed framework is a multi-agent system where each agent must learn to navigate a complex, non-convex decision space. Therefore, it is essential to first validate the collective convergence of the learning process. Fig. 2 presents the empirical training performance, plotting the maximum Q-value found across all agents at each episode. This training curve illustrates that the agents' policies rapidly improve during an initial learning phase of approximately 500 episodes, after which the system-wide maximum Q-value stabilizes. This demonstrates that the exploratory learning process successfully converges to a set of stable and effective policies, which provides the foundation for the robust results presented in the subsequent sections.

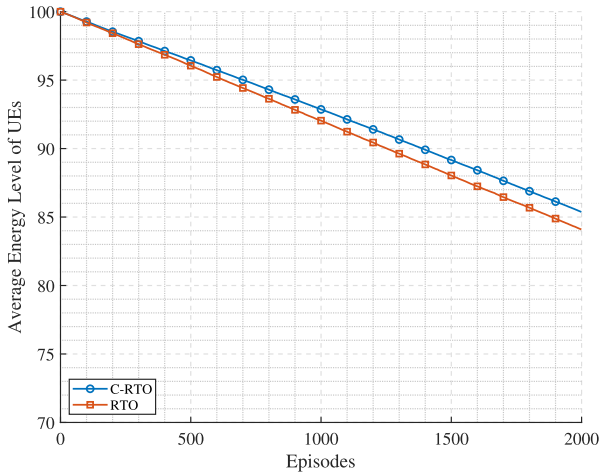

FIGURE 3. Comparison of average UE energy levels for C-RTO vs. RTO.

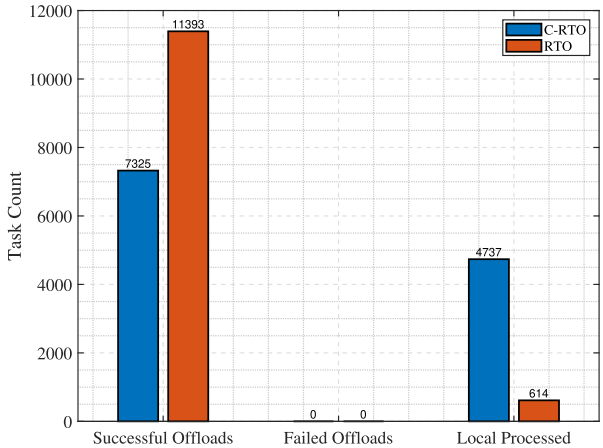

FIGURE 4. Comparison of task processing outcomes for C-RTO vs. RTO.

B. PERFORMANCE VS. COOPERATIVE MARL: THE RESILIENCY-ENERGY TRADE-OFF

To evaluate the impact of different multi-agent learning strategies as suggested by modern MARL literature, we conducted a comparative analysis of our proposed RTO algorithm against a cooperative baseline (C-RTO). The results, presented in Figs. 3 and 4, reveal that while cooperative learning can offer marginal gains in energy preservation, it does so by failing to address the primary challenge of resilient task offloading.

Fig. 3 shows that the average energy level of UEs under the RTO algorithm is only slightly lower than under C-RTO, demonstrating that our approach remains highly competitive in energy efficiency. However, this marginal difference is decisively outweighed by RTO's superior performance in achieving its primary goal of task resiliency, as shown in Fig. 4.

Fig. 4 illustrates that RTO achieves nearly double the number of successful offloads compared to C-RTO. This metric is critical, as it directly reflects the algorithm's success in navigating the complex constraints of our $(h+1)$-server permutation strategy to guarantee task resiliency. In contrast, the C-RTO baseline exhibits a clear drawback: it avoids the complexity of resilient offloading by defaulting to local execution far more frequently. This strategy, while simple and energy-conserving, fails to meet the reliability demands of mission-critical applications.

Therefore, the results validate the effectiveness of our RTO framework. The slightly higher energy expenditure is the necessary cost for achieving a far more resilient and reliable system, proving RTO's superior ability to jointly optimize for the dual objectives central to our work.

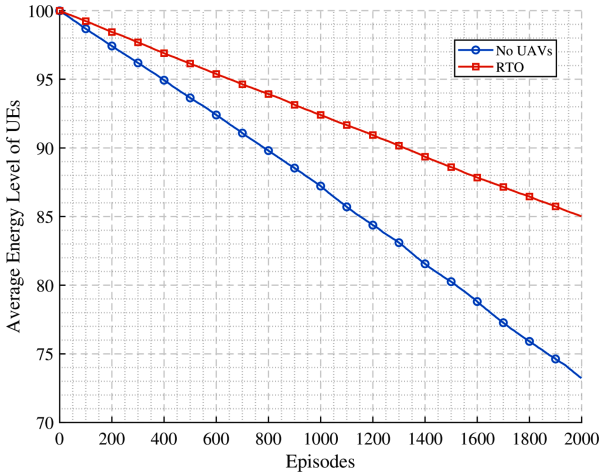

FIGURE 5. Average UE energy levels across episodes for the RTO and *No UAVs* scenarios.

C. TESTING RTO VS NO UAV-ASSISTED SYSTEM

Here we illustrate the value of using the UAVs in serving UEs carrying out their offloaded tasks and aiming to preserve their energy versus a system that lacks UAVs involvement.

1) AVERAGE UE ENERGY LEVELS ACROSS EPISODES

Fig. 5 shows the comparison of average UE energy levels over 2000 episodes for two scenarios: RTO and *No UAVs*. In the RTO scenario, UAVs assist UEs by offloading computational tasks, which leads to a significantly slower energy depletion over time. On the other hand, in the *No UAVs* scenario, UEs are forced to handle tasks either locally or offload them to BSs, resulting in faster energy consumption due to the higher energy cost of local processing. The absence of UAVs in this scenario leads to more energy-intensive operations, causing the UEs to drain their energy reserves much quicker than in the UAV-assisted scenario. This trend demonstrates the advantage of using UAVs to offload tasks and reduce the energy burden on UEs.

2) TASK OUTCOMES ACROSS EPISODES

Fig. 6 presents a comparative analysis of task execution outcomes across 2000 episodes. The bars illustrate three distinct categories of task execution: successfully offloaded tasks, failed offloading attempts, and locally processed tasks. In

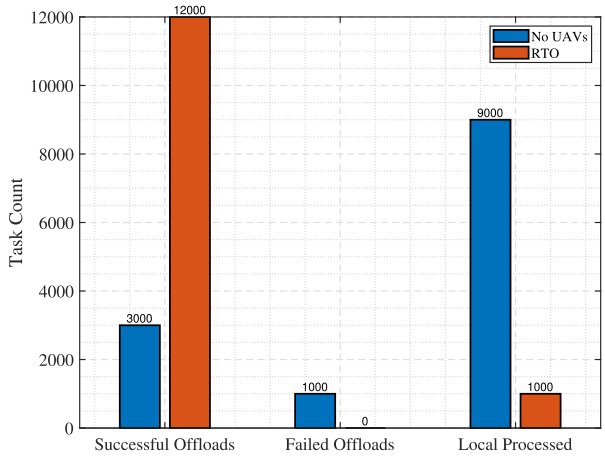

FIGURE 6. Task outcomes across episodes for the RTO and *No UAVs* scenarios.

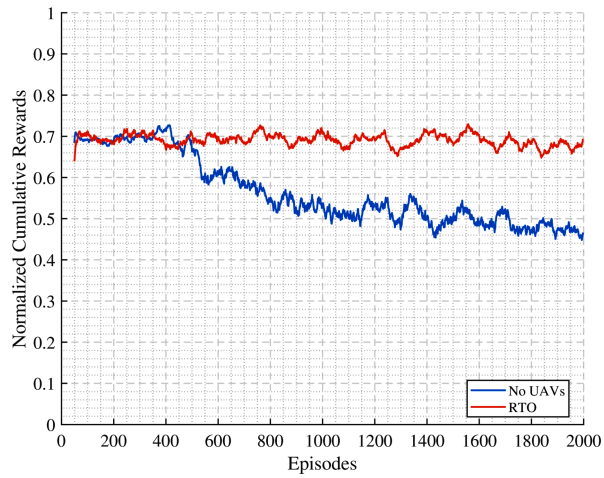

FIGURE 7. Normalized Cumulative rewards for the RTO and *No UAVs* scenarios.

the RTO scenario, the RL agent efficiently learns to conduct task allocation, resulting in a substantial increase in successful offloads while reducing the number of failed offloading attempts and local executions. The presence of UAVs enhances system flexibility by enabling dynamic task delivery to nearby BSs, thereby reducing the energy burden on UEs and ensuring higher offloading success rates. Conversely, in the *No UAVs* scenario, UEs rely exclusively on static BS for offloading, leading to a noticeable increase in failed offloads due to limited server availability and network congestion. The lack of UAV-assisted task distribution forces a greater proportion of tasks to be processed locally, significantly increasing the energy expenditure of UEs and reducing overall system efficiency.

3) NORMALIZED CUMULATIVE REWARDS

Fig. 7 illustrates the normalized cumulative rewards of UEs across simulation episodes, comparing the RTO and *No UAVs* scenarios. The RTO approach exhibits a consistent upward

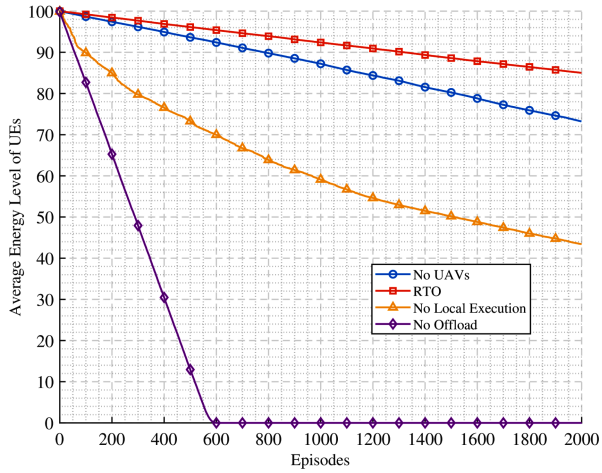

FIGURE 8. Comparison of UE average energy level for RTO with 3 benchmark algorithms (*No UAVs*, No Local Execution, and *No Offload*).

trend in cumulative rewards, indicating an effective learning process that improves task offloading efficiency over time. The RL agent adapts its policy by considering energy constraints, available offloading options, and system congestion, leading to optimal task allocation decisions. As a result, UEs in the RTO scenario experience lower energy consumption, increased successful task offloads, and reduced reliance on local execution. In contrast, the *No UAVs* scenario demonstrates significantly lower cumulative rewards, highlighting the inefficiencies associated with the absence of UAV support. Without UAVs acting as task relays, UEs struggle to offload tasks effectively, leading to higher failure rates and increased local execution, which ultimately drains their energy resources at a faster rate.

D. TESTING RTO VS SCENARIOS

Here we simulate the outcomes of testing RTO vs. multiple scenarios like *No UAVs*, *No Offload*, and *No local processing*. To ensure statistical consistency, each experimental scenario was simulated over 10 independent trials, and the results presented are averaged across these trials. The main purpose is to highlight the performance of our proposed RL-base solution, excelling over the other benchmark approaches with respect to the UE energy degradation along the different episodes and for different task sizes, the average energy level of UEs at episode 2000 (considered as a sufficient time span to achieve a stationary behavior), and the illustration of the first UE reaching zero energy.

1) COMPARISON OF AVERAGE ENERGY LEVEL OF UES

Fig. 8 illustrates the average UE energy levels over 2000 episodes for the four considered scenarios. The RTO algorithm exhibits a significant advantage over all baseline approaches, maintaining a higher average energy level across episodes. The UAV-assisted offloading mechanism ensures an optimal distribution of computational tasks, reducing the reliance on local execution and thereby preserving the energy

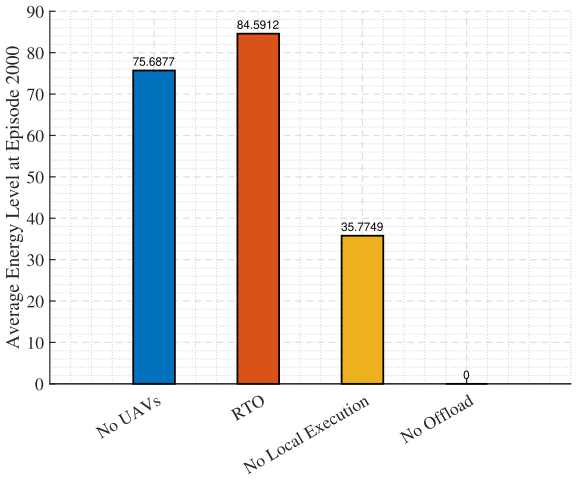

FIGURE 9. Comparison of UE average energy level at episode 2000 for RTO and 3 benchmark algorithms (*No UAVs*, No Local Execution, and *No Offload*).

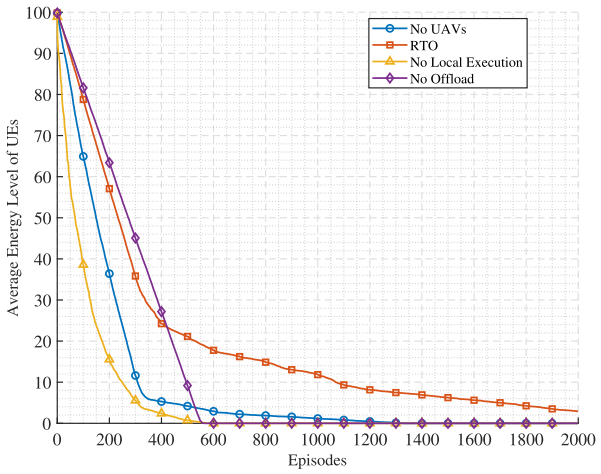

FIGURE 10. Comparison of UE time to depletion for RTO and 3 benchmark algorithms (*No UAVs*, No Local Execution, and *No Offload*).

of UEs. In contrast, the *No UAVs* and *No local processing* scenarios show a much steeper decline in energy levels, as UEs either struggle with local execution or are forced to offload to BS directly, sometimes in poor communication channel conditions, increasing transmission power consumption. The *No Offload* scenario performs the worst, as UEs must process all tasks locally, leading to the fastest depletion of energy reserves.

2) COMPARISON OF UE AVERAGE ENERGY LEVEL AT EPISODE 2000

Fig. 9 provides a snapshot of the average energy levels of UEs at the final episode of our simulation (episode 2000) for each scenario. The RTO approach achieves the highest remaining energy, confirming its effectiveness in balancing offloading decisions and conserving UE power. The *No UAVs* and *No local processing* scenarios show significantly lower final energy levels, as UEs expend more energy due to inefficient task allocation. Meanwhile, in the *No Offload* case, all UEs reach complete energy depletion by the end of the simulation, emphasizing the impracticality of purely local execution in an energy-constrained environment.

3) TIME TO TOTAL DEPLETION

Fig. 10 highlights the comparative energy depletion behavior in case of aggressive task offloading via increasing the task sizes, demonstrating how quickly each scenario forces UEs to consume their energy reserves. The *No Offload* scenario reaches zero energy for all UEs the fastest, confirming the severe inefficiency of pure local execution. The *No local processing* and *No UAVs* scenarios also show rapid energy depletion, as offloading limitations result in increased energy consumption. The RTO algorithm significantly outperforms all benchmarks, enabling UEs to sustain their energy for a longer period due to the optimal offloading strategy enabled by RL and UAV assistance.

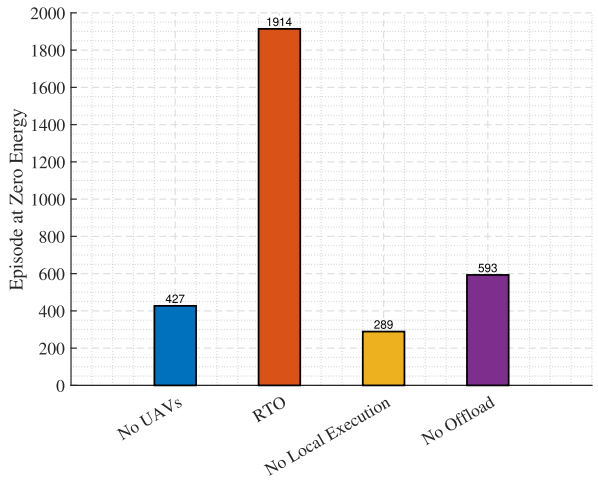

FIGURE 11. Episode for first UE depletion for RTO and 3 benchmark algorithms (*No UAVs*, No Local Execution, and *No Offload*).

4) FIRST DEPLETED UE

Finally, Fig. 11 presents a comparative analysis of the first UEs reaching zero energy across different scenarios. More precisely, we show the episode number in which the first UE reaches zero energy for each approach. The results emphasize that UEs in the *No Local Execution* scenario experience the fastest energy depletion, followed by *No Offload* and *No UAVs*. The RTO algorithm extends UE operational longevity by dynamically optimizing task allocation, significantly delaying the first instance of energy depletion among UEs.

E. SENSITIVITY MEASUREMENTS VS BENCHMARKS

In this part, we evaluate the performance of the proposed RTO algorithm under varying system conditions, such as increasing the number of UEs, UAVs, BSs, and task sizes. To provide a comprehensive performance assessment, we compare RTO with three benchmark algorithms: (1) **RAT** (RL-based Trajectory control), which applies actor-critic Deep Q-Networks

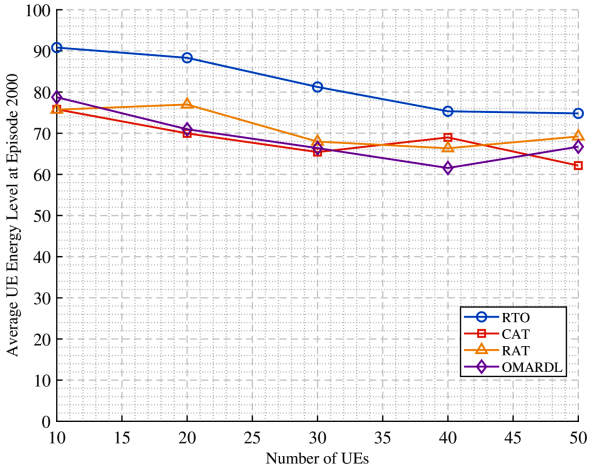

FIGURE 12. Comparison of average UE energy level at episode 2000 for RTO, CAT, RAT, and OMADRL under increasing number of UEs.

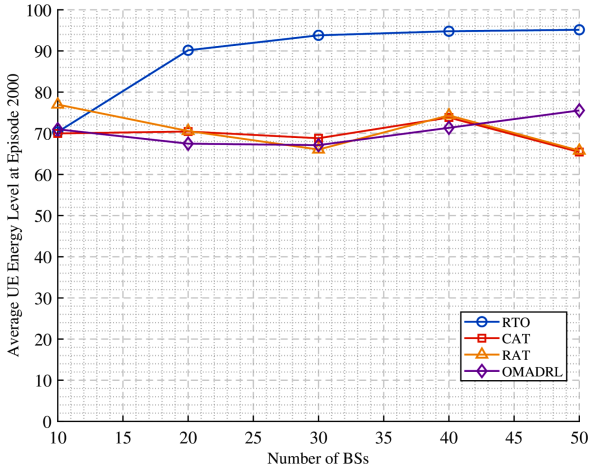

FIGURE 13. Comparison of average UE energy level at episode 2000 for RTO, CAT, RAT, and OMADRL under increasing number of BSs.

with prioritized experience replay to make real-time trajectory decisions in dynamic F-MEC environments [35]; (2) **CAT** (Convex optimization-based Trajectory control), which iteratively optimizes user association, resource allocation, and UAV trajectories using block coordinate descent methods [35]; and (3) **OMADRL** (Optimization-embedding Multi-Agent Deep Reinforcement Learning), a hybrid framework that integrates multi-agent DRL for UAV trajectory learning with embedded optimization for offloading decisions, aiming to improve convergence and energy efficiency while ensuring service fairness [36].

These particular benchmarks were selected to ensure a comprehensive comparison against a diverse set of state-of-the-art methodologies. They represent three distinct and highly relevant solution categories:

- **CAT** was chosen as a representative of traditional, non-learning optimization techniques, providing a baseline from classical methods.
- **RAT** allows for a direct comparison against another pure deep reinforcement learning approach to highlight the specific advantages of our RTO framework.
- **OMADRL** serves as a state-of-the-art benchmark representing modern hybrid methods that combine both optimization and reinforcement learning.

Evaluating RTO against this varied set of algorithms provides a robust validation of its performance and adaptability.

1) VARYING NUMBER OF USERS

Fig. 12 analyzes the impact of increasing the number of UEs on the average UE energy level at episode 2000. As the number of UEs increases from 10 to 50, all algorithms exhibit a general downward trend in energy retention, indicating higher competition for limited offloading and communication resources. However, the proposed RTO algorithm consistently outperforms the benchmark methods across all user densities. Notably, while CAT and OMADRL show a marked decline in energy levels—especially beyond 30 UEs—RTO maintains energy levels above 75% even under maximum UE load. RAT demonstrates relatively stable but moderate performance, while OMADRL suffers from early performance degradation. These results highlight the superior adaptability and robustness of RTO in managing energy-efficient task offloading and trajectory control under increased user demand.

2) VARYING NUMBER OF BSS

Fig. 13 evaluates the average UE energy level in response to increasing the number of BSs. As the BS count grows from 10 to 50, the RTO algorithm shows a clear and consistent improvement in energy efficiency, with energy levels rising from approximately 75% to over 95%. This demonstrates RTO's superior ability to leverage the expanded offloading infrastructure. In contrast, the benchmark algorithms—CAT, RAT, and OMADRL—exhibit relatively flat performance, indicating limited scalability and suboptimal BS utilization. RAT experiences an initial advantage at low BS counts but quickly plateaus, while CAT and OMADRL remain consistently below RTO across all scenarios. These results confirm that RTO can efficiently adapt to increased infrastructure capacity, achieving better energy distribution across UEs under higher BS availability.

3) VARYING NUMBER OF UAVS

Fig. 14 highlights the effect of increasing the number of UAVs on the average UE energy level at episode 2000. As the number of UAVs grows from 5 to 25, all algorithms demonstrate improvements in energy efficiency due to enhanced spatial coverage and reduced offloading congestion. However, RTO consistently maintains a significant performance margin, achieving energy levels above 90% when 20 or more UAVs are deployed. The steep improvement between 5 and 15 UAVs for RTO indicates the framework's ability to scale and coordinate task offloading effectively as aerial resources increase. In comparison, CAT and RAT follow similar but more

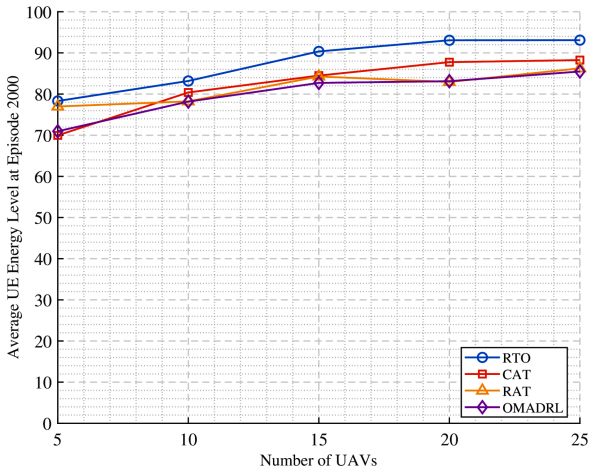

FIGURE 14. Comparison of average UE energy level at episode 2000 for RTO, CAT, RAT, and OMADRL under increasing number of UAVs.

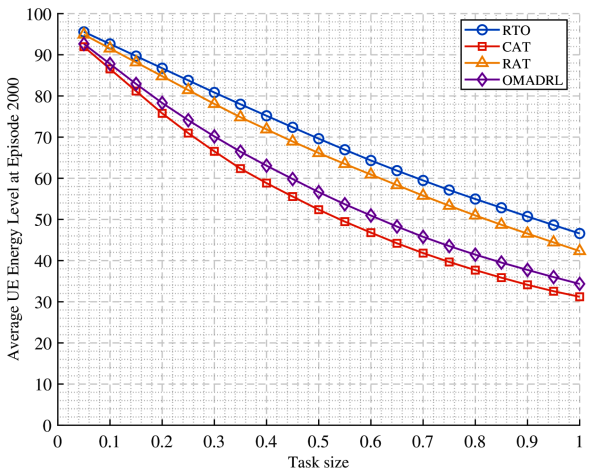

FIGURE 15. Comparison of average UE energy level at episode 2000 for RTO, CAT, RAT, and OMADRL under increasing task sizes.

modest trends, and OMADRL lags slightly behind at lower UAV densities. These results confirm that RTO leverages UAV mobility and availability more efficiently than the benchmark algorithms, resulting in superior energy conservation for user devices.

4) VARIATION IN TASK SIZE

Fig. 15 explores the impact of increasing computational task sizes on the average UE energy level at episode 2000. As task size grows from 0.01 to 1.0, all algorithms experience a steady decline in energy levels, reflecting the heavier energy burden required to process or offload larger tasks. Among the evaluated methods, RTO consistently preserves higher energy levels across the entire task size range. For instance, at high task loads (e.g., task size ≥ 0.8), RTO maintains over 50% energy on average, while CAT and OMADRL drop to the 30–40% range. RAT performs moderately well but still lags behind RTO. The performance gap is most noticeable for mid-range task sizes (0.4–0.6), where RTO achieves a significant energy efficiency advantage of more than 15–20% over the nearest competitor. These results emphasize the effectiveness of RTO's adaptive decision-making in sustaining energy efficiency under increasing computational demand.

VIII. CONCLUSION

This paper introduced an RL framework that integrates an $(h+1)$-server permutation strategy to jointly optimize for UE energy efficiency and task resiliency in UAV-assisted MEC networks. Our simulation results validated this approach, demonstrating that the RTO algorithm consistently outperforms diverse state-of-the-art benchmarks. Key findings show that our framework's performance scales effectively with more infrastructure, achieving over 95% energy efficiency with added base stations where other methods plateau. Furthermore, by preserving over 50% of UE energy under heavy computational loads—a 15–20% margin over competitors—this work provides a key insight: integrating structured resiliency with intelligent offloading is most critical and effective under network stress. These results confirm the framework's potential to deliver robust and energy-efficient services for next-generation networks.

As a natural extension, future research will investigate more realistic mobility models for both UAVs and UEs, including adaptive trajectory planning based on real-time network dynamics. We also aim to explore mobility-induced changes in user association to further improve system responsiveness.

REFERENCES

[1] F. Zhou, Y. Wu, H. Sun, and Z. Chu, "UAV-enabled mobile edge computing: Offloading optimization and trajectory design," in *Proc. IEEE Int. Conf. Commun.*, 2018, pp. 1–6.
[2] M. Mozaffari, W. Saad, M. Bennis, and M. Debbah, "Mobile unmanned aerial vehicles (UAVs) for energy-efficient Internet of Things communications," *IEEE Trans. Wireless Commun.*, vol. 16, no. 11, pp. 7574–7589, Nov. 2017.
[3] T. Ramesh et al., "A review of UAV-based applications for precision agriculture: Crop monitoring, mapping, and spraying," *Comput. Electron. Agriculture*, vol. 198, 2022, Art. no. 107098.
[4] A. Kalantari et al., "A UAV-based system for real-time video surveillance and object tracking," *J. Intell. Robotic Syst.*, vol. 100, no. 3, pp. 1065–1081, 2020.
[5] H. Wu et al., "Task offloading for mobile edge computing in vehicular networks: A deep reinforcement learning approach," *IEEE Trans. Veh. Technol.*, vol. 68, no. 2, pp. 1992–2003, Feb. 2019.
[6] Y. Xu, T. M. Khan, Y. Song, and E. Meijering, "Edge deep learning in computer vision and medical diagnostics: A comprehensive survey," *Artif. Intell. Rev.*, vol. 58, no. 3, pp. 1–78, 2025.
[7] J. Liu et al., "Reliability-enhanced task offloading in mobile edge computing environments," *IEEE Internet Things J.*, vol. 9, no. 13, pp. 10382–10396, Jul. 2022.
[8] H. Wang, H. Xu, H. Huang, M. Chen, and S. Chen, "Robust task offloading in dynamic edge computing," *IEEE Trans. Mobile Comput.*, vol. 22, no. 1, pp. 500–514, Jan. 2023.
[9] W. Zhang, S. Zeadally, H. Zhou, H. Zhang, N. Wang, and V. C. Leung, "Joint service quality control and resource allocation for service reliability maximization in edge computing," *IEEE Trans. Commun.*, vol. 71, no. 2, pp. 935–948, Feb. 2023.
[10] J. Liu and Q. Zhang, "Offloading schemes in mobile edge computing for ultra-reliable low latency communications," *IEEE Access*, vol. 6, pp. 12825–12837, 2018.
[11] J. Chen, C. Feng, D. Feng, and S. Meng, "Multi-edge computing offloading for ultra-reliable and low-latency communication," in *Proc. 13th Int. Conf. Wireless Commun. Signal Process.*, 2021, pp. 1–6.

[12] U. M. Malik, M. A. Javed, J. Frnda, and J. Nedoma, "SMRETO: Stable matching for reliable and efficient task offloading in fog-enabled IoT networks," *IEEE Access*, vol. 10, pp. 111579–111590, 2022.
[13] L. Qin, H. Lu, Y. Chen, B. Chong, and F. Wu, "Towards decentralized task offloading and resource allocation in user-centric MEC," *IEEE Trans. Mobile Comput.*, vol. 23, no. 12, pp. 11807–11823, Dec. 2024.
[14] X. Hou, Z. Ren, J. Wang, S. Zheng, and H. Zhang, "Latency and reliability oriented collaborative optimization for multi-UAV aided mobile edge computing system," in *Proc. IEEE INFOCOM IEEE Conf. Comput. Commun. Workshops*, 2020, pp. 150–156.
[15] N. Hu et al., "Building agile and resilient UAV networks based on SDN and blockchain," *IEEE Netw.*, vol. 35, no. 1, pp. 57–63, Jan./Feb., 2021.
[16] K. Peng, B. Zhao, M. Bilal, and X. Xu, "Reliability-aware computation offloading for delay-sensitive applications in MEC-enabled aerial computing," *IEEE Trans. Green Commun. Netw.*, vol. 6, no. 3, pp. 1511–1519, Sep. 2022.
[17] M. Zhao, X. Zhang, Z. Meng, and X. Hou, "Reliable DNN partitioning for UAV swarm," in *Proc. Int. Wireless Commun. Mobile Comput.*, 2022, pp. 265–270.
[18] Y. Yu, X. Bu, K. Yang, H. Yang, X. Gao, and Z. Han, "UAV-aided low latency multi-access edge computing," *IEEE Trans. Veh. Technol.*, vol. 70, no. 5, pp. 4955–4967, May 2021.
[19] A. A. Baktayan, A. T. Zahary, and I. A. Al-Baltah, "A systematic mapping study of UAV-enabled mobile edge computing for task offloading," *IEEE Access*, vol. 12, pp. 101936–101970, 2024.
[20] Z. Chen, J. Yang, and Z. Zhou, "UAV-assisted MEC offloading strategy with peak AOI boundary optimization: A method based on DDQN," *Digit. Commun. Netw.*, vol. 10, no. 6, pp. 1790–1803, 2024.
[21] P. Zhang et al., "Deep reinforcement learning based computation offloading in UAV-assisted edge computing," *Drones*, vol. 7, no. 3, 2023, Art. no. 213.
[22] A. A. Baktayan, A. T. Zahary, A. Sikora, and D. Welte, "Computational offloading into UAV swarm networks: A systematic literature review," *EURASIP J. Wireless Commun. Netw.*, vol. 2024, no. 1, 2024, Art. no. 69.
[23] J. Huang, S. Xu, J. Zhang, and Y. Wu, "Resource allocation and 3 D deployment of UAVs-assisted MEC network with air-ground cooperation," *Sensors*, vol. 22, no. 7, 2022, Art. no. 2590.
[24] Z. Hu, D. Zhou, C. Shen, T. Wang, and L. Liu, "Task offloading strategy for UAV-assisted mobile edge computing with covert transmission," *Electronics*, vol. 14, no. 3, 2025, Art. no. 446.
[25] H. Fu, J. Wang, J. Chen, P. Ren, Z. Zhang, and G. Zhao, "Dense multi-agent reinforcement learning aided multi-UAV information coverage for vehicular networks," *IEEE Internet Things J.*, vol. 11, no. 12, pp. 21274–21286, Jun. 2024.
[26] S. Du, X. Chen, L. Jiao, and Y. Lu, "Energy efficient task offloading for UAV-assisted mobile edge computing," in *Proc. 2021 China Automat. Congr.*, 2021, pp. 6567–6571.
[27] W. Liu, B. Li, W. Xie, Y. Dai, and Z. Fei, "Energy-efficient task offloading in UAV-enabled MEC via multi-agent cooperation," *IEEE Trans. Green Commun. Netw.*, vol. 7, no. 1, pp. 123–136, Jan. 2023.
[28] S. Ahmad, R. Noor, M. I. Khan, and M. Alazab, "Energy efficient UAV-enabled mobile edge computing for IoT devices: A review," *IEEE Access*, vol. 9, pp. 74129–74148, 2021.
[29] M. Zhang et al., "Energy-efficient task offloading in UAV-RIS-assisted mobile edge computing with NOMA," *IEEE Trans. Veh. Technol.*, vol. 72, no. 1, pp. 123–136, Jan. 2023.
[30] W. Liu, B. Li, W. Xie, Y. Dai, and Z. Fei, "Task offloading strategy for UAV-assisted mobile edge computing systems with covert communication," *IEEE Trans. Green Commun. Netw.*, vol. 8, no. 2, pp. 456–469, Feb. 2024.
[31] M. Li, N. Cheng, J. Gao, Y. Wang, L. Zhao, and X. Shen, "Energy-efficient UAV-assisted mobile edge computing: Resource allocation and trajectory optimization," *IEEE Trans. Veh. Technol.*, vol. 69, no. 3, pp. 3424–3438, Mar. 2020.
[32] H. Wang, W. Zhang, Y. Li, and W. Zhang, "Energy-efficient task offloading in wireless-powered MEC," *Mathematics*, vol. 12, no. 15, 2023, Art. no. 2326.
[33] J. Chen, P. Yang, S. Ren, Z. Zhao, X. Cao, and D. Wu, "Enhancing AIoT device association with task offloading in aerial MEC networks," *IEEE Internet Things J.*, vol. 11, no. 1, pp. 174–187, Jan. 2024.
[34] C. Wang, D. Zhai, R. Zhang, L. Cai, L. Liu, and M. Dong, "Joint association, trajectory, offloading, and resource optimization in air and ground cooperative MEC systems," *IEEE Trans. Veh. Technol.*, vol. 73, no. 9, pp. 13076–13089, Sep. 2024.
[35] L. Wang, K. Wang, C. Pan, W. Xu, N. Aslam, and A. Nallanathan, "Deep reinforcement learning based dynamic trajectory control for UAV-assisted mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 21, no. 10, pp. 3536–3550, Oct. 2022.
[36] X. Li, X. Du, N. Zhao, and X. Wang, "Computing over the sky: Joint UAV trajectory and task offloading scheme based on optimization-embedding multi-agent deep reinforcement learning," *IEEE Trans. Commun.*, vol. 72, no. 3, pp. 1355–1369, Mar. 2024.

MOHAMED EL-EMARY received the M.Sc. degree in electronics and communication engineering from the Arab Academy for Science Technology and Maritime Transport, Egypt, in 2018. He is currently working toward the Ph.D. degree in software and IT engineering with the École de Technologie Supérieure (ÉTS), Montreal, QC, Canada. From 2012 to 2020, he was a Network Consultant with Orange Business Services, Egypt. His current research interests include next-generation mobile communication systems, machine learning and resource management.

DIALA NABOULSI (Member, IEEE) received the Ph.D. degree in computer science from INSA Lyon, Villeurbanne, France, in 2015. She is an Associate Professor with the École de Technologie Supérieure (ÉTS), Montreal, QC, Canada. She has been involved in many Canadian and European research projects, with a strong record of industrial collaborations. Her research interests include mobile networks, virtualized networks, and wireless networks.

RAZVAN STANICA received the M.Eng. and Ph.D. degrees in computer science, from INP Toulouse, Toulouse, France, in 2008 and 2011, respectively. He is an Associate Professor with INSA Lyon, France, and a Research Scientist with the Inria Agora team of the CITI laboratory. His research interests include wireless mobile networks, with a special focus on communication networks in urban environments.