



Article

# Task Offloading and Resource Allocation for Augmented Reality Applications in UAV-Based Networks Using a Dual Network Architecture

Dat Van Anh Duong , Shathee Akter and Seokhoon Yoon \* 

Department of Electrical, Electronic and Computer Engineering, University of Ulsan, Ulsan 44610, Republic of Korea; datdva@ulsan.ac.kr (D.V.A.D.); shathee5152@ulsan.ac.kr (S.A.)

\* Correspondence: seokhoonyoon@ulsan.ac.kr; Tel.: +82-52-259-1403

**Abstract:** This paper proposes a novel UAV-based edge computing system for augmented reality (AR) applications, addressing the challenges posed by the limited resources in mobile devices. The system uses UAVs equipped with edge computing servers (UECs) specifically to enable efficient task offloading and resource allocation for AR tasks with dependent relationships. This work specifically focuses on the problem of dependent tasks in AR applications within UAV-based networks. This problem has not been thoroughly addressed in previous research. A dual network architecture-based task offloading (DNA-TO) algorithm is proposed, leveraging the DNA framework to enhance decision-making in reinforcement learning while mitigating noise. In addition, a Karush–Kuhn–Tucker-based resource allocation (KKT-RA) algorithm is proposed to optimize resource allocation. Various simulations using real-world movement data are conducted. The results indicate that our proposed algorithm outperforms existing approaches in terms of latency and energy efficiency.

**Keywords:** augmented reality; dependent tasks; task offloading; resource allocation; dual network architecture; execution latency; energy consumption



**Citation:** Van Anh Duong, D.; Akter, S.; Yoon, S. Task Offloading and Resource Allocation for Augmented Reality Applications in UAV-Based Networks Using a Dual Network Architecture. *Electronics* **2024**, *13*, 3590. <https://doi.org/10.3390/electronics13183590>

Academic Editor: Alejandro L. Borja

Received: 29 July 2024

Revised: 30 August 2024

Accepted: 6 September 2024

Published: 10 September 2024



**Copyright:** © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

Augmented reality (AR) [1] is a technology that overlays digital information onto the real world, enhancing interactions with the environment. Augmented reality is widely used in various fields such as urban planning [2], the military [3], education [4], healthcare [5], retail [6], and entertainment [7]. User equipment (UE) for running AR applications typically includes mobile devices such as glasses and smartphones [8,9]. However, the limitation on the computation resource and the energy capacity of mobile devices leads to long processing delays and high power consumption. One way to solve this problem is to use powerful computers in the cloud to handle compute-intensive tasks. But even though such computers are fast, sending lots of data back and forth over the internet can take a long time [10]. This can still cause delays when using AR, which is frustrating for users.

In order to reduce latency, studies [11,12] have investigated the use of mobile edge computing (MEC) for processing AR tasks. Placing edge servers closer to users allows devices to offload heavy processing to these nearby servers, effectively reducing the problem of slow data transfer over long distances. However, these studies assumed that MEC servers are located at the base stations (BSs) of cellular networks. This can be a problem because it limits where MEC services can be used. For example, they might not work well in areas without infrastructure.

To address these challenges, this study introduces a system model for AR applications, which uses a UAV-based network. UAVs with edge computing servers (UECs) can process intensive tasks, leading to substantial latency reduction and increased flexibility. This approach is well-suited to a variety of AR applications. For example, in critical scenarios

like natural disasters, fires, or accidents, vital real-time information can be delivered to first responders, aiding in identifying safe routes, locating victims, and highlighting hazards.

Task offloading and resource allocation need to be considered in the proposed system model. In AR applications, tasks are dependent. Specifically, the output of previous tasks is often the input for a subsequent task. However, the most recent studies [13–16] have focused on independent tasks, and only a few studies have addressed dependent tasks [17,18]. The existing approaches, while valuable for general task offloading scenarios, do not adequately capture the unique requirements of AR applications within the context of a UAV-based system. They fail to consider the specific characteristics of AR tasks, such as the sequential processing pipeline and the constraints of AR tasks. Therefore, the existing approaches are not suitable for AR applications and the proposed system model. It requires a more specialized solution that can effectively manage task offloading and resource allocation in the context of AR applications. In this work, we formulate the problem for dependent tasks in AR applications. Then, based on the proximal policy optimization with a dual network architecture (DNA) [19], we propose a DNA-based task offloading (DNA-TO) algorithm to solve the task offloading decision. A DNA addresses the higher noise level in policy learning compared to value learning in reinforcement learning by separating them into distinct networks. This separation potentially reduces overall system noise. In addition, a Karush–Kuhn–Tucker-based resource allocation (KKT-RA) algorithm is also proposed for resolving resource allocation.

To evaluate the performance of the proposed algorithms, we conducted simulations using real-world movement data from the Geolife GPS trajectory dataset [20], and then we compared the results with other methods. The results demonstrate that our algorithms outperform existing approaches.

The main contributions of this paper can be summarized as follows:

- This work considers a novel system model for AR applications using UECs to enable efficient task offloading and resource allocation.
- The challenge of dependent tasks in AR applications is addressed. Specifically, this work models the dependencies between tasks and formulates an optimization problem to determine optimal task offloading decisions and resource allocation considering these dependencies.
- This work proposes the DNA-TO algorithm for task offloading decisions, leveraging the DNA framework to mitigate noise in reinforcement learning. Additionally, we propose the KKT-RA algorithm for optimal resource allocation.
- Numerous simulations with various scenarios, using multiple performance metrics, were conducted to verify the performance of our proposed algorithms. The results show that the proposed algorithms achieved better results than existing methods.

The rest of this paper is structured as follows: Section 2 provides an overview of related work, while Section 3 describes the system model in more detail. The problem formulation is presented in Section 4, followed by problem decomposition in Section 5. Section 6 describes the algorithms in detail. Finally, evaluation results are presented in Section 7, and the paper concludes in Section 8.

## 2. Related Work

In the last few years, researchers have focused on optimizing both task offloading and resource allocation in MEC systems [11,12,21]. However, those studies assumed that MEC servers are located at cellular base stations (BSs). This makes the systems depend on an infrastructure, which can limit their flexibility and accessibility in various geographic and operational contexts.

To address these limitations, the concept of UAV-based MEC networks has recently emerged, in which UAVs equipped with edge servers function as mobile base stations [22,23]. In [22], a UAV-assisted VANET architecture to enhance vehicular network computation capabilities was proposed. This architecture uses UAVs as aerial base stations equipped with MEC servers, allowing vehicles to offload computationally intensive tasks. To optimize the

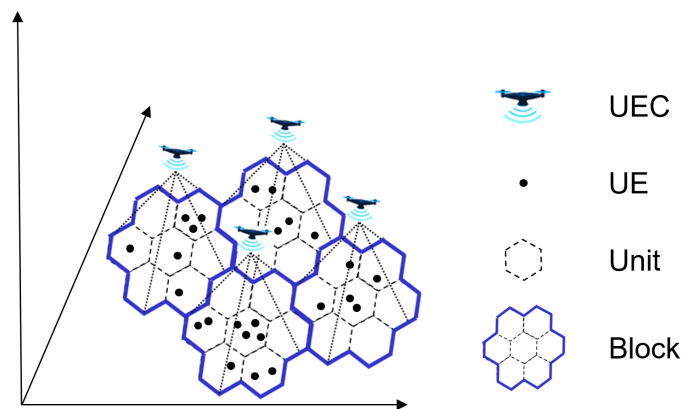
system, a joint MEC selection/resource allocation/task offloading algorithm was proposed. The goal is to minimize task processing delays by considering both the transmission time from vehicles to UAVs and the computation time on the MEC servers. The work in [23] introduced a cooperative task offloading scheme for UAV-enabled MEC systems where UEs at a distance can offload tasks to a UAV with the assistance of nearby UEs. The UAV network model consists of multiple UEs (nearby and distant) with a UAV serving as a MEC server. Each UE has a divisible computation task that can be partially offloaded to the UAV. Task offloading by distant UEs is facilitated by associated nearby UEs, reducing UAV energy consumption and ensuring reliable task offloading. The system optimizes the UAV trajectories, computations, and communication resources to minimize the weighted sum energy consumption of both UEs and the UAV. However, in these studies, dependent tasks are not considered.

A few studies have addressed dependent tasks [17,18]. In [17], the authors focused on task dependency in cooperative MEC systems where tasks are composed of multiple concatenated subtasks executed sequentially. Unlike parallel computing, these sequential tasks require the completion of one subtask before starting the next. The authors investigated two scenarios: cooperative nodes without private tasks, and cooperative nodes with their own sequential tasks. In both scenarios, optimization problems are formulated to minimize energy consumption by considering task offloading policies, communication, and computation resource allocation. In [18], task dependency in multi-user mobile edge computing systems was considered, focusing on scenarios where the input of a task on one user device requires the output of another user's task. A simplified two-user model was initially considered in order to analyze optimal solutions and the impact of task dependency on system performance. The model was then extended to a multi-user scenario, where a task by one mobile user may depend on the output from multiple mobile users. While these studies considered dependent tasks, they did not use UAV-based edge computing or address the specific characteristics of AR applications. Consequently, their findings are not directly applicable to the AR application context and our proposed system model.

Deep reinforcement learning (DRL)-based solutions have recently gained attention for addressing the challenges of task offloading and resource allocation owing to their ability to generalize and efficiently handle complex, high-dimensional problems. For example, the study in [24] proposed a meta critic method to determine the task offloading and resource allocation strategy in a digital twin (DT)-enabled UAV-assisted MEC system. In [25], DRL in a UAV-assisted multi-access edge computing network was proposed to address the challenge of task offloading and resource allocation in various scenarios. In [26], the authors proposed an approach to task offloading and resource allocation in UAV-aided MEC systems. That system model incorporates both ground MEC servers and hovering UAVs to provide low-latency communications, computations, and storage capabilities for IoT devices. The joint task offloading/load balancing/resource allocation problem was formulated as an integer optimization problem aimed at minimizing system cost. A DRL-based algorithm was studied to find an optimal offloading solution. Seid et al. [27] proposed a multi-agent deep reinforcement learning approach to optimize task offloading and resource allocation in a UAV-assisted IoT edge network. The approach considers the dynamic nature of UAV communication and resource demands to minimize computation costs while ensuring quality of service for IoT devices. However, the problem scenarios, system models, and objectives explored in previous studies are significantly different from the scenario, system model, and optimization problems addressed in our work. Consequently, these methods cannot be directly applied to our specific problem.

### 3. The System Model

A system model for processing AR applications is proposed where the area is divided into blocks, and a block is separated into units, as shown in Figure 1. The proposed system includes two key components.



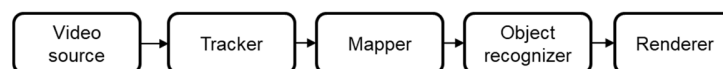
**Figure 1.** The system model for augmented reality applications.

- **User equipment (UEs):** UEs represent mobile devices capable of handling AR tasks. These tasks can be processed locally on the device or offloaded for execution on edge servers to improve performance or handle computationally intensive tasks.
- **UAVs with edge computing servers (UECs):** UECs are responsible for communicating with UEs in their designated blocks, and for processing offloaded AR tasks upon receiving requests from those UEs.

This system model can be used in many AR applications. For example, in an emergency, it can help people find a safe route to escape from dangers like fires or earthquakes. For the military, it can help find and save people who are lost or hurt. Also, it can be used to play games that mix the real world with digital images and objects.

This work studies the problems of task offloading and resource allocation for AR applications in a block with multiple UEs served by a single UEC. The set of UEs is denoted as  $\mathbb{S}_{UE}$ . For each UE  $i \in \mathbb{S}_{UE}$ , its remaining energy and available computation resource are defined as  $E_i^{UE}$  and  $F_i^{UE}$ , respectively. The UEC server also has its own resource:  $E^{UEC}$  denotes the remaining energy of the UEC, and  $F^{UEC}$  represents the available computation resource on the UEC.

Augmented reality technology seamlessly blends computer-generated content with a live video stream, creating an interactive experience where virtual elements are overlaid onto the real world. Figure 2 illustrates the processing pipeline for an AR application, which consists of five essential tasks.



**Figure 2.** Processing an AR application.

- **The video source:** This module acts as the initial stage, capturing raw video frames from the user's camera. The captured frames serve as the foundation for subsequent processing.
- **The tracker:** Playing a crucial role in determining the user's position and orientation relative to the surrounding environment, the tracker analyzes the video frames to identify and track key features, enabling accurate placement of virtual elements within the real world.
- **The mapper:** This task constructs a digital representation of the environment, often referred to as a map. By analyzing the video data and using information from the tracker, the mapper builds a model that includes feature points (details about the real world). This digital map is essential for aligning virtual objects and ensuring they interact and behave realistically within the user's physical space.
- **The object recognizer:** After analyzing video frames to identify known objects and estimating their 3D positions, this information is the input for the renderer.

- **The renderer:** The final stage of the processing pipeline, the renderer takes the processed data from previous tasks and prepares the frames for display on the user's device. This may involve overlaying virtual elements onto the video frames, incorporating information from the object recognizer, and ensuring seamless integration of the virtual content with the real world.

The five tasks in an AR application are inherently interdependent. The output from one task often serves as the input for a subsequent task in the processing pipeline. For instance, the tracker's output becomes the input for the mapper in order to build an accurate digital map. We denote the set of task types as  $\mathbb{S}_{task\_type} = \{n_1, n_2, \dots, n_5\}$ , where  $n_1, n_2, \dots, n_5$  represent the video source, tracker, mapper, object recognizer, and renderer, respectively. It is important to note that real-time processing requirements of the video source and renderer ( $n_1$  and  $n_5$ ) must be executed locally on the user's device. However, the remaining tasks ( $n_2, n_3$ , and  $n_4$ ) can be offloaded for execution on a UEC for potentially faster or more efficient processing, particularly when dealing with computationally intensive tasks.  $\mathbb{S}_t^{task}$  denotes the set of tasks to be processed at time  $t$ . For each  $l \in \mathbb{S}_t^{task}$ ,  $n(l)$ ,  $c(l)$ , and  $d(l)$  denote the task type, computation cost, and input data size of task  $l$ , respectively. In the context of AR applications, the task offloading and resource allocation problem aims to determine the optimal execution strategy for tasks. We need to determine whether to execute a task locally on the user's device or offload it to the UEC. This decision considers factors such as task complexity, resource constraints on the device, and the UEC server. Once the execution locations for tasks are determined, we need to allocate the computation resource of the UEC for offloaded tasks. The task offloading decision and the resource allocation strategy are the keys to optimizing the overall performance of the AR application to minimize energy consumption and task completion time. The definitions of notations in the system model are in Table 1.

**Table 1.** Parameters in the system model.

For the UEC	
$E^{UEC}$	The remaining energy of the UEC
$F^{UEC}$	The available computation resource on the UEC
$f_l$	The computation resource assigned to task $l$ by the UEC
$\mu^{UEC}$	The energy consumed per CPU cycle of the UEC
$p_{UEC}^R$	The power consumption per time unit of the UEC when receiving data
$h$	The height of the UEC
For the UEs	
$\mathbb{S}_{UE}$	The set of UEs
$E_i^{UE}$	The remaining energy of UE $i$
$p_i^T$	The transmission power of UE $i$
$r_i$	The uplink transmission rate from UE $i$ to the UEC
$F_i^{UE}$	The computation resource of UE $i$
$\mu_i^{UE}$	The energy consumed per CPU cycle of UE $i$

**Table 1.** *Cont.*

Other parameters	
$\mathbb{S}_{task\_type}$	The set of task types
$\mathbb{S}_i^{task}$	The set of tasks in time slot $t$
$n(l)$	The type of task $l$
$c(l)$	The computation cost of task $l$
$d(l)$	The input data size of task $l$
$a$ and $b$	Constants that depend on the propagation environment in Equation (1)
$c$	The speed of light
$B$	The sub-channel bandwidth
$t^{Loc}(l)$	The completion time of task $l$ when executing locally
$e^{Loc}(l)$	The energy consumption by UE $i$ for locally processed task $l$
$t_i^{T-Input}(l)$	The time delay for transmitting the input of task $l$ from UE $_i$ to the UEC
$e_i^{T-Input}(l)$	The energy consumption by UE $i$ when uploading task $l$ to the UEC
$e_{UEC}^{R-Input}(l)$	The energy consumption by the UEC when receiving input data for task $l$
$t_{UEC}^{Exe}(l)$	The execution time of task $l$ when executed by the UEC
$e_{UEC}^{Exe}(l)$	The energy consumption when executing task $l$ at the UEC
$t^{UEC}(l)$	The completion time of task $l$ when offloaded to the UEC
$e^{UEC}(l)$	The total energy consumption when offloading task $l$ to the UEC

### 3.1. The Communication Model

This subsection describes the communication model between UEs and the UEC. UEs connect to the UEC for task processing. The output data size after task execution on the UEC is typically much smaller than the raw input data captured by the UEs. Additionally, downlink data rates (UEC to UE) are generally higher than uplink rates. Therefore, we focus on the uplink data transfer (UE to UEC) and do not consider transferring processed data back to UEs. Similar to previous studies like [28–30], a classic air-to-ground probabilistic path loss model [31] is used. There are line-of-sight (LoS) links and non-line-of-sight (NLoS) links for communications between the UEs and the UEC. Path loss considers both free-space path loss and additional loss due to environmental factors.  $PL$  denotes path loss, calculated using the following equation:

$$PL = \frac{\eta_{LoS} - \eta_{NLoS}}{1 + a \times \exp(-b(\arctan(\frac{h}{d}) - a))} + 10 \log(h^2 + d^2) + 20 \log(f) + 20 \log(\frac{4\pi}{c}) + \eta_{NLoS} \quad (1)$$

where  $\eta_{LoS}$  and  $\eta_{NLoS}$  are the average additional losses of the free propagation space loss under LoS and NLoS conditions, respectively;  $a$  and  $b$  are constants that depend on the propagation environment (e.g., in urban [31],  $a = 9.61$ ,  $b = 0.16$ );  $h$  is the height of the UEC and  $d$  is the distance between a UE and the UEC's projection on the ground, while  $f$  is the carrier frequency, and  $c$  is the speed of light.

Let  $p_i^T$  denote the transmission power of UE  $i$ . The uplink transmission rate from UE  $i$  to the UEC is defined as  $r_i$ . Based on the calculated path loss and noise power during transmission,  $r_i$  can be determined as follows:

$$r_i = B \log_2 \left( 1 + \frac{p_i^T}{PL \times \sigma} \right) \quad (2)$$

where  $B$  is the sub-channel bandwidth and  $\sigma$  is noise power. This communication model allows us to determine the uplink transmission rate between UEs and the UEC.

### 3.2. The Computation Model

In this section, the computation model is discussed. A task can be executed locally or computed by the UEC. Therefore, processing tasks locally and computing tasks in the UEC are presented.

#### 3.2.1. Local Computing

Let  $l \in \mathbb{S}_i^{task}$  be a task of UE  $i$  at time  $t$ . Note that  $F_i^{UE}$  and  $c(l)$  are the computation resource of UE  $i$  and the computation cost of task  $l$ , respectively. When  $l$  is processed locally on UE  $i$ , the completion time is denoted  $t^{Loc}(l)$  and is calculated as follows:

$$t^{Loc}(l) = \frac{c(l)}{F_i^{UE}} \quad (3)$$

Furthermore, the energy consumed per CPU cycle of UE  $i$  is denoted  $\mu_i^{UE}$ . Let  $e^{Loc}(l)$  denote the energy consumption for local processing, so  $e^{Loc}(l)$  is computed as

$$e^{Loc}(l) = c(l) \mu_i^{UE} \quad (4)$$

#### 3.2.2. Computing in a UEC

This subsection discusses how to process a task on the UEC. Let  $l \in \mathbb{S}_i^{task}$  be the task of UE  $i$  at time  $t$ . To execute task  $l$  at the UEC, we first consider transmission of the input data for task  $l$  from UE  $i$  to the UEC. The size of the input data for task  $l$  is denoted  $d(l)$ . Let  $t_i^{T-Input}(l)$  denote the time delay for input data transmission, calculated as

$$t_i^{T-Input}(l) = \frac{d(l)}{r_i} \quad (5)$$

The energy consumption for uploading the input data from UE  $i$  is denoted as  $e_i^{T-Input}(l)$ , given by

$$e_i^{T-Input}(l) = t_i^{T-Input}(l) \times p_i^T \quad (6)$$

Let  $e_{UEC}^{R-Input}(l)$  denote the energy consumption when receiving the input data at the UEC. It is calculated as follows:

$$e_{UEC}^{R-Input}(l) = t_i^{T-Input}(l) \times p_{UEC}^R \quad (7)$$

where  $p_{UEC}^R$  represents the power consumption per time unit of the UEC when receiving data, measured in watts (W).

The execution time and the energy consumption when executing task  $l$  at the UEC are denoted  $t_{UEC}^{Exe}(l)$  and  $e_{UEC}^{Exe}(l)$ , respectively. Those values are calculated using Equations (8) and (9):

$$t_{UEC}^{Exe}(l) = \frac{c(l)}{f_l} \quad (8)$$

$$e_{UEC}^{Exe}(l) = c(l) \times \mu^{UEC} \quad (9)$$

In these equations,  $f_l$  is the computation resource assigned to task  $l$  by the UEC, and  $\mu^{UEC}$  represents the energy consumed per CPU cycle of the UEC, measured in joules per cycle (J/cycle).

To calculate the completion time and total energy consumption for task  $l$  processed on the UEC, we need to consider both input data transmission and task execution. It is important to note that tasks in AR applications are dependent. If two adjacent tasks are executed on the UEC, the input for the second task can be directly obtained from the output and input of the first task, eliminating the need for additional input data transfer from the UE to the UEC.

Let  $v$  be the task of UE  $i$  at time  $t - 1$ . We define a binary variable  $b_l$  to indicate whether task  $l$  is the subsequent task of task  $v$  in the processing pipeline and whether both tasks are executed on the UEC. If task  $l$  is the subsequent task of task  $v$  and both are processed on the UEC,  $b_l = 0$ ; otherwise,  $b_l = 1$ . Let  $t^{UEC}(l)$  denote the completion time and  $e^{UEC}(l)$  be the total energy consumption. Considering the dependency, those values are calculated with Equation (10) and Equation (11), respectively:

$$t^{UEC}(l) = b_l \times t_i^{T\_input}(l) + t_{UEC}^{Exe}(l) \quad (10)$$

$$e^{UEC}(l) = b_l \times (e_i^{T\_Input}(l) + e_{UEC}^{R\_Input}(l)) + e_{UEC}^{Exe}(l) \quad (11)$$

From Equations (10) and (11), when  $b_l = 0$ , this indicates that task  $l$  is the subsequent task of task  $v$ , and both are executed on the UEC. Consequently, the transmission time and energy consumption for sending and receiving input data from UE  $i$  to the UEC can be ignored.

#### 4. Problem Formulation

This section formulates the optimal problem of task offloading and resource allocation based on the system model described previously.

Let  $o_l$  be a binary variable at time  $t$  that indicates the task offloading decision for task  $l$ . A value of 1 means that task  $l$  is offloaded to the UEC, whereas 0 indicates local execution:

$$o_l = \begin{cases} 1, & \text{if task } l \text{ is offloaded to the UEC} \\ 0, & \text{otherwise} \end{cases} \quad (12)$$

The set of all offloading decision variables at time  $t$  is denoted by  $\mathbb{O} = \{o_l | l \in \mathbb{S}_t^{task}\}$ . Note that the resource allocation variable for task  $l$ , denoted  $f_l$ , represents the amount of computational resource allocated to task  $l$  by the UEC. For all tasks at time  $t$ , the set of computation resource variables is  $\mathbb{F} = \{f_l | l \in \mathbb{S}_t^{task}\}$ .

Let  $T$  be the total completion time of all tasks at time  $t$ , calculated as

$$T = \sum_{l \in \mathbb{S}_t^{task}} (1 - o_l) \times t^{Loc}(l) + o_l \times t^{UEC}(l) \quad (13)$$

The total energy consumption of all UEs and the UEC is denoted  $E$ . It is calculated using the following equation:

$$E = \sum_{l \in \mathbb{S}_t^{task}} (1 - o_l) \times e^{Loc}(l) + o_l \times e^{UEC}(l) \quad (14)$$

The task offloading and resource allocation problem is formulated as an optimization problem to minimize the total completion time and total energy consumption. We first define the objective function as follows:

$$J = \alpha \frac{T}{w_T \times |\mathbb{S}_t^{task}|} + (1 - \alpha) \frac{E}{w_E \times |\mathbb{S}_t^{task}|} \quad (15)$$



where a tunable parameter,  $\alpha \in [0, 1]$ , modifies the balance between minimizing completion time and minimizing energy consumption. Higher values for  $\alpha$  prioritize a faster completion time, while lower values prioritize lower energy consumption.  $|\mathbb{S}_i^{task}|$  is the total number of tasks at time  $t$ , while  $w_T$  and  $w_E$  are the coefficients for rescaling the time and energy, respectively. Specifically,  $w_T$  is the longest possible completion time for locally executing a task, and  $w_E$  is the highest energy consumption from processing a task locally. These values help to normalize the contributions of time and energy to the objective function.

The problem can be formulated as an optimization problem minimizing  $J$ :

$$\min_{\mathbb{O}, \mathbb{F}}(J) \tag{16}$$

subject to:

$$o_l \in \{0, 1\}, \forall l \in \{\mathbb{S}_i^{task} | n(l) \notin \{n_1, n_5\}\} \tag{17}$$

$$o_l = 0, \forall l \in \{\mathbb{S}_i^{task} | n(l) \in \{n_1, n_5\}\} \tag{18}$$

$$\sum_{l \in \mathbb{S}_i^{task}} (1 - o_l) \times e^{Loc}(l) + o_l \times b_l \times e_i^{T\_input}(l) \leq E_i^{UE}, \forall i \in \mathbb{S}_{UE} \tag{19}$$

$$\sum_{l \in \mathbb{S}_i^{task}} o_l \times e^{UEC}(l) - b_l \times e_i^{T\_input}(l) \leq E^{UEC} \tag{20}$$

$$f_l = 0, \forall l \in \{\mathbb{S}_i^{task} | o_l = 0\} \tag{21}$$

$$f_l > 0, \forall l \in \{\mathbb{S}_i^{task} | o_l = 1\} \tag{22}$$

$$\sum_{l \in \mathbb{S}_i^{task}} o_l \times f_l \leq F^{UEC} \tag{23}$$

Equations (17) and (18) imply that the video source and renderer can only be executed locally, whereas other types of tasks could be offloaded to the UEC or processed locally. The energy constraints of UEs and the UEC are given in Equation (19) and Equation (20), respectively. Specifically, the total energy consumption by each UE and by the UEC should not exceed their residual energy levels. Equation (21) states that the UEC does not assign the resource to locally executed tasks. By contrast, Equation (22) guarantees that if a task is offloaded and needs to be executed, the UEC must assign computing resource to the task. The limitation on the computation resource at the UEC is given in constraint (23), i.e., the total computation resource assigned to tasks must not exceed the computation resource of the UEC.

### 5. Problem Decomposition

From the problem formulation, we obtain a problem that contains two types of variables (i.e., binary variables ( $\mathbb{O}$ ) and continuous variables ( $\mathbb{F}$ )). Combining binary and continuous variables leads to a non-convex mixed-integer non-linear programming problem, which is difficult to solve and has high computational complexity. To address this problem, we decompose the problem into two simpler subproblems. The first subproblem, an integer problem, focuses on determining the optimal offloading decisions for tasks using binary variables. The second subproblem, a continuous problem, allocates resources to offload tasks using continuous variables.

In order to decompose the problem, we reformulate it as a two-stage optimization problem. Accordingly, the reformulation is as follows:

$$(P1) \quad \min_{\mathbb{O}} \left( \min_{\mathbb{F}}(J) \right) \tag{24}$$

subject to: Equations (17)–(23)

where reformulated problem P1 can be characterized as a two-stage minimization process. The first stage minimizes the objective function with respect to the decision variables ( $\mathbb{O}$ ). However, this minimization is achieved by solving a nested minimization problem for a fixed value of  $\mathbb{O}$ . This nested problem focuses on resource allocation decisions denoted by  $\mathbb{F}$  and aims to find the minimum value of  $J$ . All the original problem constraints remain.

A key observation from reformulated problem P1 is decoupling of the constraints related to task offloading decisions and resource allocation decisions. This separation allows us to analyze the problem characteristics based on the fixed values of either  $\mathbb{O}$  or  $\mathbb{F}$ . Specifically, if resource allocation  $\mathbb{F}$  is fixed, P1 becomes an integer optimization problem of the decision variables in  $\mathbb{O}$ . In contrast, if we fix the task offloading decisions ( $\mathbb{O}$ ), P1 transforms into a continuous optimization problem since minimization is performed only with respect to the continuous variables in  $\mathbb{F}$ . Taking advantage of this decoupling, we decompose the problem into two subproblems: a master problem and a subproblem. Let us denote the master problem as P2 and the subproblem as P3. The master problem can be written as follows:

$$(P2) \quad \min_{\mathbb{O}}(J^*) \tag{25}$$

subject to: Equations (17)–(20)

where  $J^*$  represents the optimal objective value obtained from subproblem P3 for a given fixed task offloading decision. P3 is as follows:

$$(P3) \quad J^* = \min_{\mathbb{F}}(J) \tag{26}$$

subject to: Equations (21)–(23)

P2 focuses on finding the optimal value of  $J^*$  with respect to the task offloading decisions ( $\mathbb{O}$ ). P3 aims to minimize original objective function  $J$  with respect to resource allocation decisions  $\mathbb{F}$ , considering the fixed task offloading decisions provided by the master problem. The constraints for the master problem include Equations (17)–(20), whereas subproblem P3 is subject to the remaining constraints, Equations (21)–(23).

### 6. Algorithm

This section presents the dual network architecture-based task offloading algorithm and the Karush–Kuhn–Tucker-based resource allocation algorithm. The proposed framework is shown in Figure 3. Specifically, DNA-TO minimizes the objective, and the best task-offloading decision is obtained. The reward of each DNA-TO solution is calculated based on the resource allocation solution from KKT-RA. The remainder of this section describes DNA-TO and KKT-RA in detail.

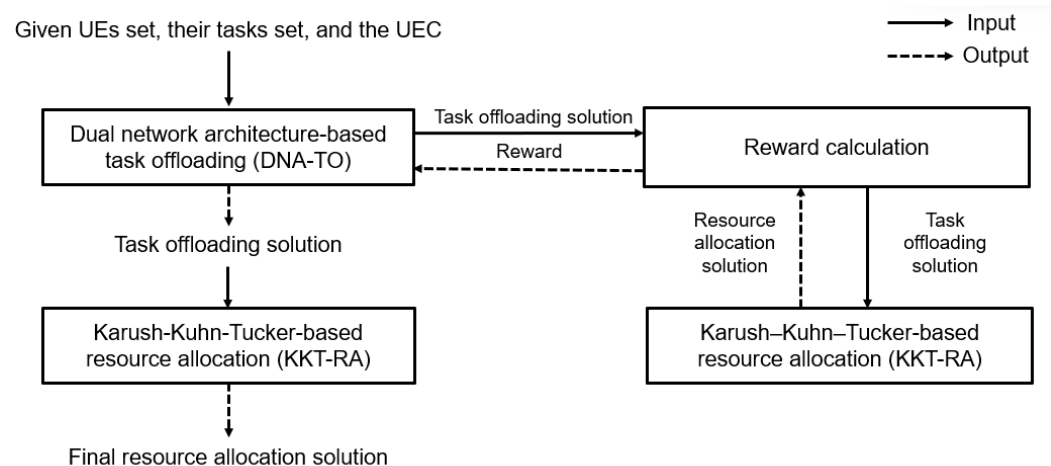


Figure 3. The framework of the proposed scheme.

### 6.1. DNA-Based Task Offloading

Proximal policy optimization with a dual network architecture [19] is a novel approach to the task offloading problem. The DNA addresses a key challenge in reinforcement learning: the presence of higher noise levels in policy learning compared to value learning. By separating these tasks into distinct networks, the DNA potentially reduces overall noise in the system. In this section, we propose the DNA-TO algorithm to solve problem P2.

#### 6.1.1. Markov Decision Process Formulation

In reinforcement learning, a model-free Markov decision process (MDP) [32] describes environments where an agent interacts and learns without a pre-defined model of transition probabilities and rewards. By exploring actions and observing the resulting states and the corresponding rewards, the agent builds its understanding of the environment. These experiences enable the agent to estimate transition probabilities and rewards. This section proposes a specific model-free MDP formulation for problem P2. Specifically, the state space, the action space, and the reward function are defined. This enables the DNA agent to learn optimal task-offloading strategies. Let  $\mathcal{S}$ ,  $\mathcal{A}$ , and  $\mathcal{R}$  denote the state space, the action space, and the reward function, respectively. Their definitions are as follows.

- (a) State space  $\mathcal{S}$ : At the beginning of each time slot  $t$ , the state that includes the environment information is collected by the UEC. Let  $s_t \in \mathcal{S}$  be the state in time slot  $t$ . It is defined as follows:

$$s_t = \{\mathbb{O}, \mathbb{P}, \mathbb{M}, \mathbb{C}, \mathbb{D}, \mathbb{G}\} \quad (27)$$

where  $\mathbb{O}$ ,  $\mathbb{P}$ ,  $\mathbb{M}$ ,  $\mathbb{C}$ ,  $\mathbb{D}$ ,  $\mathbb{G}$  are defined as follows.

- $\mathbb{O} = \{o_i \mid \forall i \in \mathbb{S}_t^{\text{task}}\}$  represents the set of offloading decisions made for tasks in time slot  $t$ . The value of  $o_i$  indicates whether task  $i$  is executed locally ( $o_i = 0$ ) or offloaded and processed by the UEC ( $o_i = 1$ ).
  - $\mathbb{P} = \{p_i \mid \forall i \in \mathbb{S}_{t-1}^{\text{task}}\}$  is the set of offloading decisions for tasks in the previous time slot ( $t - 1$ ).
  - $\mathbb{M} = \{m_i \mid \forall i \in \mathbb{S}_{UE}\}$  represents the set of relationships between tasks in time slot  $t$  and the tasks in time slot ( $t - 1$ ) for all UEs. Specifically,  $m_i$  captures the relationship between the tasks of UE  $i$ . A value of  $m_i = 1$  indicates that the current task of UE  $i$  is the subsequent task in the processing pipeline of its previous task; otherwise,  $m_i = 0$ .
  - $\mathbb{C} = \{c_i \mid \forall i \in \mathbb{S}_t^{\text{task}}\}$  represents the set of computation costs for tasks in the current time slot; the values are normalized to fall between 0 and 1.
  - $\mathbb{D} = \{d_i \mid \forall i \in \mathbb{S}_t^{\text{task}}\}$  is the set of input sizes for all tasks in the current time slot. The values are also normalized to fall between 0 and 1.
  - $\mathbb{G} = \{x_i, y_i \mid \forall i \in \mathbb{S}_t^{UE}\}$  is the set of location coordinates for all UEs in the system;  $(x_i, y_i)$  is the location of UE  $i$ . Both  $x$  and  $y$  coordinates are normalized to fall between 0 and 1.
- (b) Action space  $\mathcal{A}$ : This defines the set of all possible actions the agent can take in a single time slot. Let  $a_t = \{i \mid i \in [1, 2 \times |\mathbb{S}_t^{\text{task}}|]\}$  be the action in time slot  $t$ , where  $|\mathbb{S}_t^{\text{task}}|$  is the number of tasks in time slot  $t$ . Each action is represented by a single integer value within the range  $[1, 2 \times |\mathbb{S}_t^{\text{task}}|]$ . The value of  $a_t$  determines whether a task is offloaded or executed locally.
- Offloading: If  $a_t \leq |\mathbb{S}_t^{\text{task}}|$ , it means the task with index  $a_t$  in  $\mathbb{S}_t^{\text{task}}$  was chosen for offloading to the UEC.
  - Local execution: If  $a_t > |\mathbb{S}_t^{\text{task}}|$ , it indicates the task with index  $(a_t - |\mathbb{S}_t^{\text{task}}|)$  in  $\mathbb{S}_t^{\text{task}}$  is executed locally on the device.
- (c) Immediate reward function  $\mathcal{R}$ : To minimize the objective,  $\mathcal{R}$  is designed to negatively correlate with the objective value such that actions leading to lower objective values receive higher rewards. Additionally, to satisfy constraints (19) and (20) in P2, two penalty terms are considered. For constraint (19), let  $v_i$  be a binary variable

indicating whether the remaining energy of UE  $i$  violates constraint (19) or not. It is defined as follows:

$$v_i = \begin{cases} 1, & \text{if UE } i \text{ violates constraint (19)} \\ 0, & \text{otherwise} \end{cases} \quad (28)$$

We define  $\omega_1$  as representing the number of UEs that violate constraint (19). It is calculated as follows:

$$\omega_1 = \sum_{i \in \mathbb{S}_{UE}} v_i \quad (29)$$

For constraint (20),  $\omega_2$  is a binary variable to indicate whether constraint (20) is violated or not.  $\omega_2$  is defined as:

$$\omega_2 = \begin{cases} 1, & \text{if constraint (20) is violated} \\ 0, & \text{otherwise} \end{cases} \quad (30)$$

As penalty terms,  $\omega_1$  and  $\omega_2$  are used to discourage the agent from taking actions that violate constraints (19) and (20), respectively. Let  $s$ ,  $a$ , and  $s'$  be the current state, the action taken, and the next state, respectively.  $J(s')$  denotes the value of the objective function for the next state. The reward denoted by  $r(s, a, s')$ , received by the agent for taking action  $a$  in state  $s$  and transitioning to state  $s'$ , is calculated as follows:

$$r(s, a, s') = -[\phi \times J(s') + (1 - \phi)[\rho \frac{\omega_1}{|\mathbb{S}_{UE}|} + (1 - \rho)\omega_2]] \quad (31)$$

where  $\phi$  and  $\rho$  are two tunable parameters;  $\phi$  balances the contribution of the objective value and the penalties to the overall reward. A higher  $\phi$  emphasizes the objective, whereas a lower  $\phi$  gives more weight to the penalties, and  $\rho$  is used to adjust the effect of the two penalty terms.  $|\mathbb{S}_{UE}|$  denotes the total number of UEs and is used to normalize the value of  $\omega_1$ .

### 6.1.2. The DNA-TO Algorithm

A DNA allows for independent learning of value and policy, preventing interference. We propose a modified version of the DNA for the task offloading problem (DNA-TO). These are two networks in DNA-TO. The policy network, denoted by  $\theta_\pi(s)$ , outputs policy  $\pi$  and an estimated value,  $V_\pi(s)$ , while the value network, denoted by  $\theta_V$ , outputs only a value,  $V_V(s)$ . To improve control over noise levels for both policy and value learning, the DNA uses two separate return estimations, both using  $TD(\lambda)$  [33] as follows:

$$V_{adv}(s) = TD^{(\gamma, \lambda_\pi)}(s) \quad (32)$$

and

$$V_{targ}(s) = TD^{(\gamma, \lambda_V)}(s) \quad (33)$$

where  $\gamma \in [0, 1]$  is a discount factor, while  $\lambda_V$  and  $\lambda_\pi$  are hyperparameters that determine the balance between the variance-bias trade-off in each estimate.  $V_{targ}(s)$  is the target for training the value function, while the other value estimate,  $V_{adv}(s)$ , is used to compute advantage estimates for policy updates as follows:

$$\hat{A}(s) = V_{adv}(s) - V_V(s) \quad (34)$$

The DNA training process is divided into three distinct phases: policy, value, and distillation. Each phase focuses on optimizing a single objective using a unique optimizer and a hyperparameter set for a specific number of epochs.

- **Policy phase:** The policy phase optimizes the  $\theta_\pi$  network. It uses the clipped surrogate objective from proximal policy optimization (PPO) [34] with an entropy bonus, and the policy loss is defined as

$$L^{CLIP}(\theta_\pi) = \hat{\mathbb{E}}[\min(\varphi(\theta_\pi)\hat{A}(s), \text{clip}(\varphi(\theta_\pi), 1 - \epsilon, 1 + \epsilon)\hat{A}(s)) + c_{eb} \cdot S[\pi(s)]] \quad (35)$$

where  $\varphi(\theta_\pi)$  is the probability ratio between current and old policies, and  $\epsilon$  is the clipping coefficient.  $S[\pi(s)]$  and  $c_{eb}$  are the entropy of the policy and the entropy bonus coefficient, respectively.

- **Value phase:** The value phase optimizes the  $\theta_V$  network. It uses the squared-error value loss. The loss function is defined as follows:

$$L^{VF}(\theta_V) = \hat{\mathbb{E}}[(V_V(s) - V_{targ}(s))^2] \quad (36)$$

- **Distillation phase:** In this phase, knowledge from value network  $\theta_V$  is transferred to the policy network,  $\theta_\pi$ , through a constrained distillation update [35]. This is achieved using the mean squared error while softly constraining the network’s policy. The distillation loss function is

$$L^D(\theta_\pi) = \hat{\mathbb{E}}[(V_\pi(s) - V_V(s))^2] + \delta \cdot \hat{\mathbb{E}}[\text{KL}(\pi_{\text{old}}(\cdot|s), \pi(\cdot|s))] \quad (37)$$

where  $\delta$  is the policy constraint coefficient.  $\text{KL}(\pi_{\text{old}}(\cdot|s), \pi(\cdot|s))$  is the Kullback–Leibler (KL) divergence between the old and new policy distributions.

### 6.2. Karush–Kuhn–Tucker-Based Resource Allocation

In this section, resource allocation problem P3 is solved by using Karush–Kuhn–Tucker conditions [36]. As shown in constraint (21), if task  $l$  is locally executed,  $f_l = 0$ . Therefore, only tasks that are offloaded and executed at the UEC must be assigned computation resources. Let  $\mathbb{U}_t^{\text{task}}$  be the set of tasks that are offloaded and executed at the UEC in time slot  $t$ . Then, the formulation of resource allocation problem P3 can be reformulated as follows:

$$(P4) \quad \min_{\mathbb{F}} \left( \frac{\alpha}{w_T} \sum_{l \in \mathbb{U}_t^{\text{task}}} \frac{c(l)}{f_l} \right) \quad (38)$$

subject to:

$$f_l > 0, \forall l \in \mathbb{U}_t^{\text{task}} \quad (39)$$

$$\sum_{l \in \mathbb{U}_t^{\text{task}}} f_l \leq F^{UEC} \quad (40)$$

Constraints (39) and (40) ensure that each task in  $\mathbb{U}_t^{\text{task}}$  receives computation resources, while also ensuring that the total allocated resources do not exceed the UEC’s computational capacity. The constraints in Equations (39) and (40) are convex. Let us denote the objective function of P4 as  $\Gamma(\mathbb{F})$ . Its second-order derivatives with respect to  $f_l$  are calculated and obtained as follows:

$$\frac{\delta^2 \Gamma(\mathbb{F})}{\delta f_l^2} = \frac{\alpha}{w_T} \times \frac{2}{f_l^3} > 0, \quad \forall l \in \mathbb{U}_t^{\text{task}} \quad (41)$$

$$\frac{\delta^2 \Gamma(\mathbb{F})}{\delta f_l \delta f_{l'}} = 0, \quad \forall l, l' \in \mathbb{U}_t^{\text{task}} \text{ and } l \neq l' \quad (42)$$

Equations (41) and (42) demonstrate that the Hessian matrix of  $\Gamma(\mathbb{F})$  is positive-definite. Combined with the convex of the constraints in Equations (39) and (40), this indicates that P4 is a convex optimization problem, making it suitable for optimization using KKT

conditions. Let  $f_l^*$  represent the optimal resource allocation solution and let  $\Gamma(\mathbb{F}^*)$  be the corresponding optimal objective function value. By applying the KKT conditions, we obtain  $f_l^*$  and  $\Gamma(\mathbb{F}^*)$  as follows:

$$f_l^* = \frac{F^{UEC} \sqrt{c(l)}}{\sum_{l \in \mathbb{U}_t^{\text{task}}} \sqrt{c(l)}} \quad (43)$$

$$\Gamma(\mathbb{F}^*) = \frac{\alpha}{w_T} \times \frac{1}{F^{UEC}} \left[ \sum_{l \in \mathbb{U}_t^{\text{task}}} \sqrt{c(l)} \right]^2 \quad (44)$$

### 6.3. Algorithm Complexity

This section analyzes the complexity of the proposed algorithms for real-time decision-making in task offloading and resource allocation. In DNA-TO, a trained policy network with a fixed architecture drives task offloading decisions at each time slot  $t$ . Initially, a greedy algorithm, with time complexity  $O(|\mathbb{S}_t^{\text{task}}|)$ , provides an initial task offloading solution. Then, for  $N$  iterations, the state is input to the policy network to obtain task offloading decisions with time complexity  $O(N)$ . The time complexity of the DNA-TO algorithm is  $O(|\mathbb{S}_t^{\text{task}}| + N)$ . The KKT-RA algorithm, responsible for resource allocation, has a time complexity of  $O(|\mathbb{S}_t^{\text{task}}|)$ . The combined time complexity of the proposed approach is  $O(|\mathbb{S}_t^{\text{task}}| + N)$ .

For memory requirements, the primary memory requirement for the DNA-TO algorithm comes from loading the policy network into memory. In this work, the policy network has a feedforward architecture consisting of  $L$  hidden layers, each containing  $N$  neurons. Analyzing the policy network architecture, let  $M_\theta$  be the memory requirement (in bytes) for loading the network. It can be estimated as follows:

$$M_\theta = 4[|\mathbb{S}_t^{\text{task}}|(8N + 1) + N^2(L - 1) + N(L + 1) + 1] \quad (45)$$

Let  $M_{RA}$  denote the memory requirement (in bytes) for KKT-RA. It can be estimated as

$$M_{RA} = 28|\mathbb{S}_t^{\text{task}}| + 4 \quad (46)$$

The total memory requirements of the proposed approach can be estimated as  $(M_\theta + M_{RA})$  bytes. Overall, the proposed approach shows a relatively low complexity, making it suitable for real-time scenarios.

## 7. Evaluation Results

This section evaluates the performance of the proposed algorithm through simulations conducted with various parameter configurations. The results obtained are then compared to other methods.

A UAV-based network is used for the simulation scenario. As shown in Figure 4, a block in the simulation area comprises 169 units. Each unit is a hexagon with a side length of 5 m. A UEC is responsible for communication and task offloading within this block. The UEC is positioned at the center of its block, with its height randomly set between 70 and 100 m. This ensures communication with all UEs in the block at a reasonable speed. The Geolife GPS trajectory dataset was selected to simulate UE movements due to its inclusion of both human location and time information within a suitably sized collection area, which is crucial for accurately simulating AR applications in our scenario. The dataset contains the movement trajectories of 182 individuals in Beijing, China, collected over five years. An area with a high density of data points within the dataset was selected and designated as the simulation area. The Geolife GPS dataset has sparse data for each day, so we combined one month of data (April 2009) to represent a single day in our simulation. Specifically, the original date information was disregarded, and the trajectories of a person

on different days were treated as those of different people in the combined dataset. We extracted the latitude and longitude of UEs from this processed data, setting the altitude to zero. Finally, we randomly selected data from 20 people for the single UEC scenario and 60 people for the multiple UEC scenario. The path loss model parameters ( $a$ ,  $b$ ,  $\mu_{LoS}$ ,  $\mu_{NLoS}$ ) were set to 9.61, 0.16, 1.0, and 20, respectively, following the setup for urban environments in [31]. These values characterize the signal attenuation expected in an urban setting, influencing the received signal strength and the reliability of the communication links between UEs and the UEC. The carrier frequency,  $f$ , is set to 2 GHz. This frequency is a popular frequency range for various wireless communication technologies and is widely used in various studies [16,37,38]. It can support the high data rates required for AR applications. The noise power,  $\sigma$ , from UEs to the UEC is set to  $-100$  dBm. This represents a relatively low level of background noise, which is a reasonable assumption for UAV-based AR networks operating in open environments. The bandwidth,  $B$ , of each UE is set to 20 MHz. This bandwidth allocation aligns with common practices in modern mobile networks and enables sufficient data rates for AR content transmission. The transmission power of UEs is set to 20 dBm, a realistic value for mobile devices and commonly used in various studies [16,39,40]. The tunable parameter,  $\alpha$ , is set to 0.5. This means the impacts of execution latency and energy consumption are equally important in the objective function. For  $\epsilon$ ,  $c_{cb}$ , and  $\delta$ , the values were set to 0.2,  $10^{-3}$ , and 1, respectively, as in [19]. The available processing capacity of the UEC was set to 50 GHz (Intel Core i9-13900K), while UE capacities were randomly assigned within the range of 1.8 GHz to 2 GHz (ARM Cortex-A57). Following [13,41], the thermal design power of each processor was used to estimate energy consumption. Specifically, energy consumption per CPU cycle ( $\mu^{UEC}$  and  $\mu_i^{UE}$ ) was set to  $1.1 \times 10^{-9}$  W and  $2.259 \times 10^{-9}$  W for UEs and UECs, respectively. Additional experimental settings are detailed in Table 2. In the table, the data sizes for tasks and the required CPU cycles were set up based on the input sizes and the required CPU cycles for AR applications [11,42–44].

The simulations were run on a computer equipped with an Intel Core i9-10900X CPU @3.70 GHz with an NVIDIA Quadro RTX 4000 GPU and 64 GB of RAM.

To see how well the proposed algorithms performed, we compared simulation results against three baseline methods.

- Greedy algorithm: This method selects the processing device with the lowest weighted sum of energy consumption and execution latency for each task. The KKT-RA algorithm, the same algorithm in our proposal, was used to calculate these metrics and determine the final resource allocation solution.
- Genetic algorithm (GA): A modified genetic algorithm [45] determined the task offloading decision, while the KKT-RA algorithm was used to compute the objective function value and the final resource allocation solution.
- Soft actor critic (SAC): This approach is similar to DNA-TO. However, instead of using the DNA for task offloading, a soft actor critic [46] determines the task offloading decision. Like other methods, KKT-RA was used to calculate the reward for each state and find the final resource allocation solution.

The performance of the proposed algorithm and the baseline algorithms was evaluated in two scenarios: a UAV-based network with a single UEC and a UAV-based network with multiple UECs (i.e., three UECs). The key performance metrics (i.e., average task execution latency and total energy consumption) were collected over 600 time slots (each time slot is one second) and used to compare the different methods.

**Table 2.** Experimental settings.

Parameter	Value
Number of UECs	{1, 3}
Number of UEs	20
Bandwidth $B$ (in megahertz)	20
Data size of task (in megabytes):	
Video source	[0.5, 8]
Tracker	[0.5, 8]
Mapper	[0.5, 16]
Object Recognizer	[0.5, 8]
Renderer	[0.5, 8]
Required CPU cycles to complete task:	
Video source	$[0.1 \times 10^9, 0.5 \times 10^9]$
Tracker	$[0.1 \times 10^9, 0.5 \times 10^9]$
Mapper	$[0.1 \times 10^9, 1 \times 10^9]$
Object Recognizer	$[0.1 \times 10^9, 0.5 \times 10^9]$
Renderer	$[0.1 \times 10^9, 0.5 \times 10^9]$
Energy capacity of UECs (in megajoules)	6
Energy capacity of UEs (in kilojoules)	[10, 20]
Computation resource of UECs (in gigahertz)	50
Computation resource of UEs (in gigahertz)	[1.8, 2]
The carrier frequency $f$ (in gigahertz)	2
The transmission power of UEs (in decibel-milliwatt)	20
Path loss parameter $a$	9.61
Path loss parameter $b$	0.16
Path loss parameter $\mu_{LoS}$	1.0
Path loss parameter $\mu_{NLoS}$	20
Tunable parameter $\alpha$	0.5
The clipping coefficient $\epsilon$	0.2
The entropy bonus coefficient $c_{cb}$	$10^{-3}$
The policy constraint coefficient $\delta$	1

### 7.1. Simulation Results

#### 7.1.1. Results with a Single UEC

This section focuses on evaluating the proposed algorithm in a scenario with a single UEC, as shown in Figure 4. The impact of varying the number of UEs, the task sizes, and the required CPU cycles is presented and analyzed.

##### (a) The effect from different numbers of UEs

Figure 5a shows the impact on average task execution latency from varying the number of UEs. As the number of UEs increased from 5 to 20, a general trend of increasing latency was observed across the algorithms. The greedy algorithm exhibited the highest latency. DNA-TO demonstrated the best overall performance, particularly in scenarios with a larger number of UEs (e.g., 20 UEs). Notably, when the number of UEs was relatively low (e.g., 5, 10, and 15 UEs), the performance gap between DNA-TO, the GA, and the SAC was small. However, this gap widened significantly at 20 UEs, indicating that DNA-TO is able to handle more complex environments, while the GA and the SAC only performed comparably well in simpler scenarios.

The relationship between the number of UEs and the total energy consumption is presented in Figure 5b. In general, the upward trend in energy consumption was observed



for all algorithms when the number of UEs increased. The greedy algorithm and the GA exhibited similar energy consumption, both of which were higher than the other methods. DNA-TO also obtained the best performance with a larger number of UEs. While the performance gap between DNA-TO and the SAC algorithm was minor with fewer UEs, it became more pronounced at 20 UEs, further demonstrating DNA-TO’s ability to efficiently manage task offloading decisions in complex scenarios with higher user densities.

(b) The effect of different task data sizes

In this section, results from various task data sizes are presented for 20 UEs. Table 3 shows the different data size setups, which increased from setup 1 to setup 4 for each task.

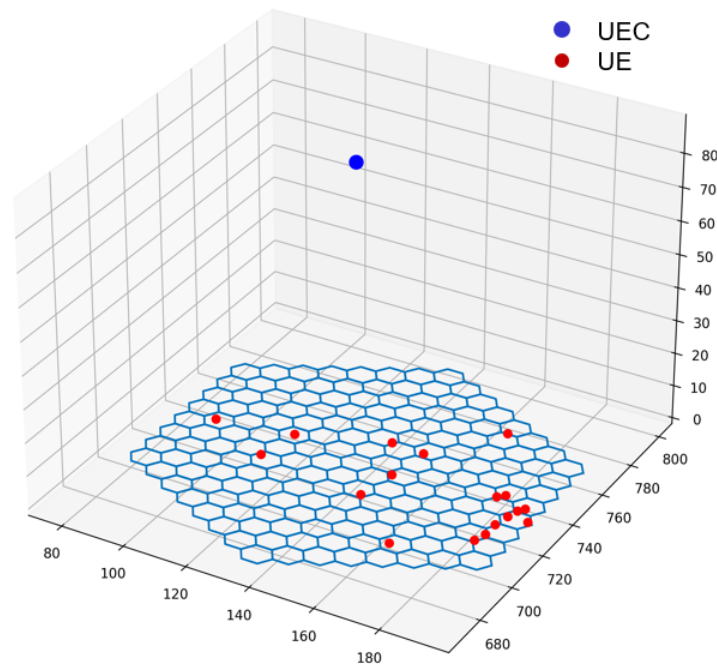


Figure 4. The scenario with a single UEC.

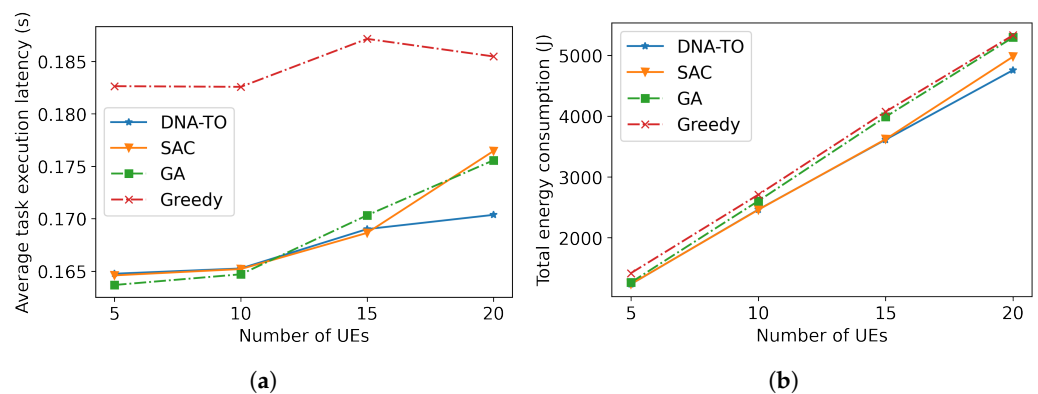


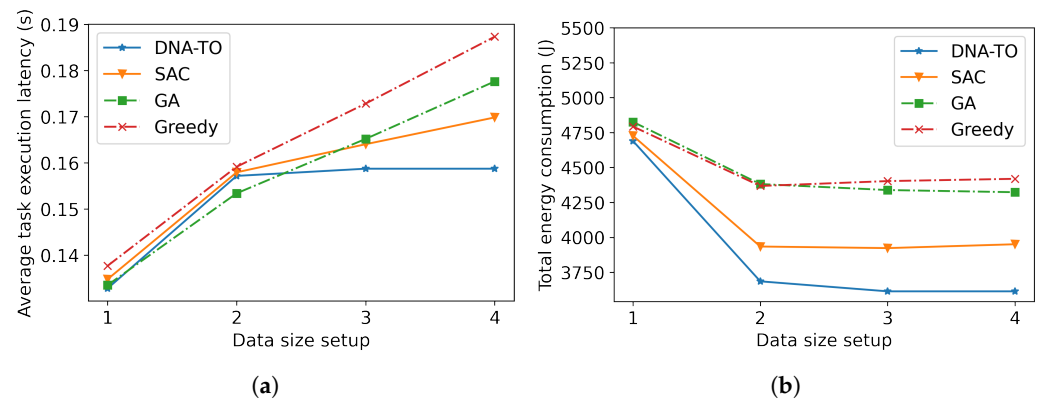
Figure 5. The results from varying the number of UEs. (a) Average task execution latency. (b) Total energy consumption.

The average execution latency with different task data sizes is shown in Figure 6a. Latency increased as data size increased, but the rate of increase varied among the algorithms, with the GA and the greedy algorithm exhibiting a significantly steeper incline compared to DNA-TO and the SAC. DNA-TO had lower latency, highlighting its efficiency in handling tasks with varying complexities. The gap between DNA-TO and other methods widened as data size increased, and DNA-TO was able to maintain low latency even with larger data sizes.

**Table 3.** Different task data size setups (in megabytes).

Task	Setup 1	Setup 2	Setup 3	Setup 4
Video source	[0.5, 2]	[2, 4]	[4, 6]	[6, 8]
Tracker	[0.5, 2]	[2, 4]	[4, 6]	[6, 8]
Mapper	[0.5, 4]	[2, 8]	[4, 12]	[6, 16]
object Recognizer	[0.5, 2]	[2, 4]	[4, 6]	[6, 8]
Renderer	[0.5, 2]	[2, 4]	[4, 6]	[6, 8]

Figure 6b shows total energy consumption across different task data sizes. The trend shows a general decrease in total energy consumption as the data size increased for all algorithms. Observe that the GA and the greedy algorithm exhibited the highest energy consumption, whereas DNA-TO consistently showed the lowest. The energy difference between DNA-TO and the other algorithms became more significant with larger data sizes, highlighting its efficiency in energy consumption, especially for larger data sizes.



**Figure 6.** Results of the various task data sizes. (a) Average task execution latency. (b) Total energy consumption.

(c) The effect of different CPU cycle requirements

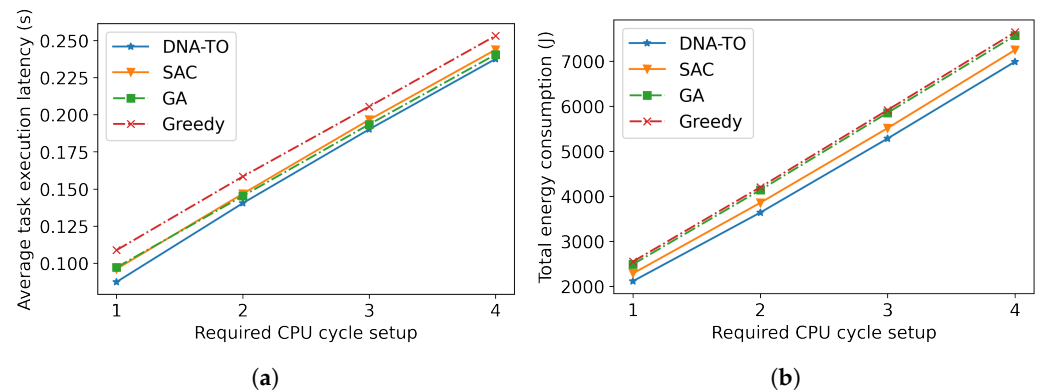
In this section, the results when the required CPU cycles varied were collected for 20 UEs. Table 4 presents the different requirements. Needed CPU cycles increased from setup 1 to setup 4 for each task.

**Table 4.** CPU cycle requirements.

Task	Setup 1	Setup 2	Setup 3	Setup 4
Video source	$[0.1 \times 10^9, 0.2 \times 10^9]$	$[0.2 \times 10^9, 0.3 \times 10^9]$	$[0.3 \times 10^9, 0.4 \times 10^9]$	$[0.4 \times 10^9, 0.5 \times 10^9]$
Tracker	$[0.1 \times 10^9, 0.2 \times 10^9]$	$[0.2 \times 10^9, 0.3 \times 10^9]$	$[0.3 \times 10^9, 0.4 \times 10^9]$	$[0.4 \times 10^9, 0.5 \times 10^9]$
Mapper	$[0.1 \times 10^9, 0.4 \times 10^9]$	$[0.2 \times 10^9, 0.6 \times 10^9]$	$[0.3 \times 10^9, 0.8 \times 10^9]$	$[0.4 \times 10^9, 1 \times 10^9]$
Object Recognizer	$[0.1 \times 10^9, 0.2 \times 10^9]$	$[0.2 \times 10^9, 0.3 \times 10^9]$	$[0.3 \times 10^9, 0.4 \times 10^9]$	$[0.4 \times 10^9, 0.5 \times 10^9]$
Renderer	$[0.1 \times 10^9, 0.2 \times 10^9]$	$[0.2 \times 10^9, 0.3 \times 10^9]$	$[0.3 \times 10^9, 0.4 \times 10^9]$	$[0.4 \times 10^9, 0.5 \times 10^9]$

Figure 7a illustrates the relationship between average task execution latency and the required CPU cycles. As the required CPU cycles increased, there was a general upward trend in latency. The greedy algorithm showed the most significant increase in latency, followed by the GA, the SAC, and DNA-TO. DNA-TO obtained the lowest latency for all CPU cycle setups.

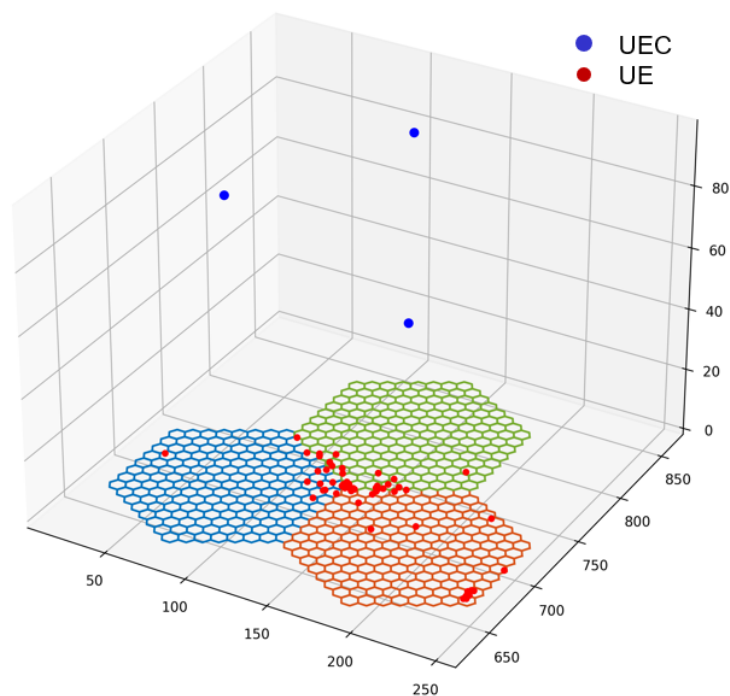
Total energy consumption is displayed in Figure 7b. In general, energy consumption increased as the required number of CPU cycles increased. DNA-TO maintained the lowest energy consumption across all CPU cycle requirements. This shows DNA-TO can optimize energy usage more effectively than the other methods.



**Figure 7.** The results from the various CPU cycle requirements. (a) Average task execution latency. (b) Total energy consumption.

### 7.1.2. Results with Multiple UECs

As shown in Figure 8, this section evaluates the proposed algorithm in a scenario with three UECs. Each UEC was responsible for 20 UEs in each block. The effects from varying the task sizes and the required CPU cycles are investigated and analyzed. For the different task sizes, the configurations detailed in Table 3 were employed, while varying the CPU cycle requirements used configurations specified in Table 4.



**Figure 8.** The scenario with three UECs.

(a) The effect from different task data sizes

Figure 9a displays the average task execution latency for different data size setups. It shows a trend similar to the single UEC scenario. When the data size increased, latency increased, indicating that larger data sizes lead to longer processing times. DNA-TO outperformed the other algorithms, maintaining the lowest average latency with larger data sizes (e.g., setups 3 and 4). This demonstrates that DNA-TO schedules tasks effectively, even with large data sizes.

Figure 9b presents the total energy consumption of the four different algorithms (DNA-TO, SAC, GA, and greedy) across the various data size setups. Overall, total energy consumption decreased when data size increased. The lowest energy consumption was obtained by DNA-TO, while the GA archived the highest value. This indicates DNA-TO can effectively minimize energy consumption while maintaining performance.

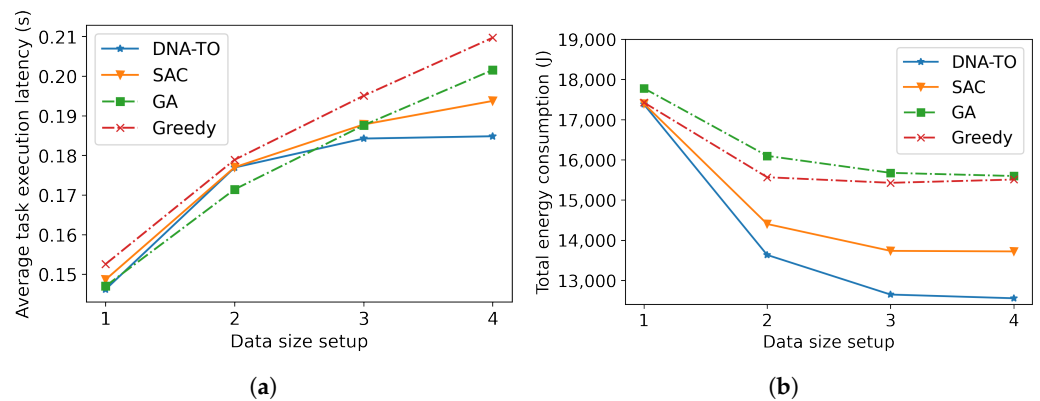


Figure 9. Results from the various task data sizes. (a) Average task execution latency. (b) Total energy consumption.

(b) The effect of different CPU cycle requirements

The relationship between average task execution latency and different required CPU cycles is presented in Figure 10a. As the required CPU cycles increased, there was a clear upward trend in latency for all algorithms. DNA-TO obtained the shortest latency, while the SAC and the GA had similar latency, and the greedy algorithm had the longest. This demonstrates DNA-TO's ability to maintain shorter latency than the others, even under high computational loads.

Figure 10b presents the total energy consumption for the four algorithms with different CPU cycle requirements. DNA-TO had the lowest energy consumption, while the SAC consumed slightly more energy than DNA-TO. Both DNA-TO and the SAC algorithm remained significantly more efficient than the GA and the greedy algorithm.

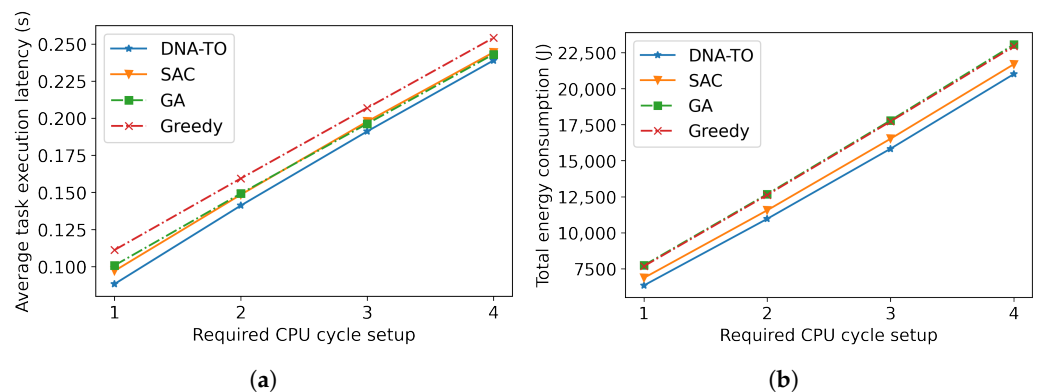


Figure 10. The results from the various CPU cycle requirements. (a) Average task execution latency. (b) Total energy consumption.

The proposed system operates independently on each UEC, handling the UEs within its designated block. This decentralized approach ensures that increasing the number of UECs does not affect the algorithm's performance on any individual UEC. Consequently, the proposed system can readily scale to accommodate networks with a larger number of UECs. This scalability is demonstrated in the simulation results, where the performance in terms of latency remains consistent between scenarios with one UEC and three UECs. The observed increase in total energy consumption is expected due to the corresponding rise in the number of UECs, UEs, and processed tasks.

## 8. Conclusions

This paper presents a novel system model and algorithms for task offloading and resource allocation in UAV-based augmented reality networks. By equipping UAVs with edge computing servers, we overcame the limitations of traditional mobile devices and cloud-based solutions for AR applications. Our model explicitly addressed the challenges of dependent tasks in AR. An optimization problem was formulated to minimize the overall task completion time and energy consumption, subject to constraints on resource availability. To resolve these issues, we proposed the DNA-TO algorithm, leveraging the advantages of a dual network architecture to reduce noise and enhance decision-making in the reinforcement learning process. Additionally, the KKT-RA algorithm efficiently allocated computation resources to offloaded tasks. To evaluate the effectiveness of our proposed approach, we conducted various simulations using real movement data from the Geolife GPS trajectory dataset. The results show that our algorithm outperformed existing methods (e.g., the SAC algorithm, the GA, and the greedy algorithm) in terms of both latency and energy consumption. However, there are several potential challenges to implementing the proposed method in real-world scenarios. Factors such as adverse weather conditions, varying obstacle distributions, and limited UEC battery capacity can significantly impact the feasibility and quality of the AR experience. In future work, we plan to analyze the effects of these challenges and extend this work to study solutions, such as strategies for UEC charging and replacement, cooperative operation strategies between UECs, and using UECs with diverse capabilities to improve the AR experience. Additionally, we intend to test our proposed framework in real-world AR applications to validate its performance under realistic conditions and identify further areas for improvement.

**Author Contributions:** Conceptualization, D.V.A.D., S.A., and S.Y.; Formal analysis, D.V.A.D., S.A., and S.Y.; Funding acquisition, S.Y.; Investigation, D.V.A.D., S.A., and S.Y.; Methodology, D.V.A.D., S.A., and S.Y.; Project administration, S.Y.; Supervision, S.Y.; Validation, D.V.A.D., S.A., and S.Y.; Writing—original draft, D.V.A.D., S.A., and S.Y.; Writing—review and editing, D.V.A.D., S.A., and S.Y. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work was supported by the 2024 Research Fund of University of Ulsan.

**Data Availability Statement:** Dataset available on request from the authors.

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

## References

1. Mendoza-Ramírez, C.E.; Tudon-Martínez, J.C.; Félix-Herrán, L.C.; Lozoya-Santos, J.d.J.; Vargas-Martínez, A. Augmented reality: Survey. *Appl. Sci.* **2023**, *13*, 10491. [[CrossRef](#)]
2. Boos, U.C.; Reichenbacher, T.; Kiefer, P.; Sailer, C. An augmented reality study for public participation in urban planning. *J. Locat. Based Serv.* **2023**, *17*, 48–77. [[CrossRef](#)]
3. Mao, C.C.; Chen, C.H. Augmented reality of 3D content application in common operational picture training system for army. *Int. J. Hum.-Comput. Interact.* **2021**, *37*, 1899–1915. [[CrossRef](#)]
4. Al-Ansi, A.M.; Jaboob, M.; Garad, A.; Al-Ansi, A. Analyzing augmented reality (AR) and virtual reality (VR) recent development in education. *Soc. Sci. Humanit. Open* **2023**, *8*, 100532. [[CrossRef](#)]
5. Ashwini, K.B.; Savitha, R.; Ananya, H. Application of Augmented Reality Technology for Home Healthcare Product Visualization. *ECS Trans.* **2022**, *107*, 10921. [[CrossRef](#)]

6. Davis, L.; Aslam, U. Analyzing consumer expectations and experiences of Augmented Reality (AR) apps in the fashion retail sector. *J. Retail. Consum. Serv.* **2024**, *76*, 103577.
7. Villagran-Vizcarra, D.C.; Luviano-Cruz, D.; Pérez-Domínguez, L.A.; Méndez-González, L.C.; Garcia-Luna, F. Applications analyses, challenges and development of augmented reality in education, industry, marketing, medicine, and entertainment. *Appl. Sci.* **2023**, *13*, 2766. [\[CrossRef\]](#)
8. Huynh, L.N.; Lee, Y.; Balan, R.K. Deepmon: Mobile gpu-based deep learning framework for continuous vision applications. In Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services, Niagara Falls, NY, USA, 19–23 June 2017; pp. 82–95.
9. Liu, Q.; Huang, S.; Opadere, J.; Han, T. An edge network orchestrator for mobile augmented reality. In Proceedings of the IEEE INFOCOM 2018-IEEE Conference on Computer Communications, Honolulu, HI, USA, 16–19 April 2018; pp. 756–764.
10. Dustdar, S.; Avasalcai, C.; Murturi, I. Invited Paper: Edge and Fog Computing: Vision and Research Challenges. In Proceedings of the 2019 IEEE International Conference on Service-Oriented System Engineering (SOSE), San Francisco, CA, USA, 4–9 April 2019; pp. 96–9609. [\[CrossRef\]](#)
11. Wang, C.; Zhang, S.; Qian, Z.; Xiao, M.; Wu, J.; Ye, B.; Lu, S. Joint server assignment and resource management for edge-based MAR system. *IEEE/ACM Trans. Netw.* **2020**, *28*, 2378–2391. [\[CrossRef\]](#)
12. Chen, X.; Liu, G. Joint optimization of task offloading and resource allocation via deep reinforcement learning for augmented reality in mobile edge network. In Proceedings of the 2020 IEEE International Conference on Edge Computing (EDGE), Beijing, China, 19–23 October 2020; pp. 76–82.
13. Akter, S.; Duong, D.V.A.; Kim, D.Y.; Yoon, S. Task Offloading and Resource Allocation in UAV-aided Emergency response Operations via Soft Actor Critic. *IEEE Access* **2024**, *12*, 69258–69275. [\[CrossRef\]](#)
14. Sun, G.; He, L.; Sun, Z.; Wu, Q.; Liang, S.; Li, J.; Niyato, D.; Leung, V.C.M. Joint Task Offloading and Resource Allocation in Aerial-Terrestrial UAV Networks with Edge and Fog Computing for Post-Disaster Rescue. *IEEE Trans. Mob. Comput.* **2024**, *23*, 8582–8600. [\[CrossRef\]](#)
15. Morshed Alam, M.; Moh, S. Joint Optimization of Trajectory Control, Task Offloading, and Resource Allocation in Air-Ground Integrated Networks. *IEEE Internet Things J.* **2024**, *11*, 24273–24288. [\[CrossRef\]](#)
16. Wu, G.; Liu, Z.; Fan, M.; Wu, K. Joint Task Offloading and Resource Allocation in Multi-UAV Multi-Server Systems: An Attention-based Deep Reinforcement Learning Approach. *IEEE Trans. Veh. Technol.* **2024**, *73*, 11964–11978. [\[CrossRef\]](#)
17. An, X.; Fan, R.; Hu, H.; Zhang, N.; Atapattu, S.; Tsiftsis, T.A. Joint Task Offloading and Resource Allocation for IoT Edge Computing with Sequential Task Dependency. *IEEE Internet Things J.* **2022**, *9*, 16546–16561. [\[CrossRef\]](#)
18. Yan, J.; Bi, S.; Zhang, Y.J.; Tao, M. Optimal Task Offloading and Resource Allocation in Mobile-Edge Computing with Inter-User Task Dependency. *IEEE Trans. Wirel. Commun.* **2020**, *19*, 235–250. [\[CrossRef\]](#)
19. Aitchison, M.; Sweetser, P. DNA: Proximal policy optimization with a dual network architecture. *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 35921–35932.
20. Zheng, Y.; Fu, H.; Xie, X.; Ma, W.Y.; Li, Q. *Geolife GPS Trajectory Dataset—User Guide*, Geolife GPS trajectories 1.1 ed.; Microsoft, Redmond, WA, USA, 2011.
21. Fan, R.; Liang, B.; Zuo, S.; Hu, H.; Jiang, H.; Zhang, N. Robust Task Offloading and Resource Allocation in Mobile Edge Computing with Uncertain Distribution of Computation Burden. *IEEE Trans. Commun.* **2023**, *71*, 4283–4299. [\[CrossRef\]](#)
22. He, Y.; Zhai, D.; Zhang, R.; Du, J.; Aujla, G.S.; Cao, H. A Mobile Edge Computing Framework for Task Offloading and Resource Allocation in UAV-assisted VANETs. In Proceedings of the IEEE INFOCOM 2021-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Vancouver, BC, Canada, 10–13 May 2021; pp. 1–6. [\[CrossRef\]](#)
23. Xu, D.; Xu, D. Cooperative task offloading and resource allocation for UAV-enabled mobile edge computing systems. *Comput. Netw.* **2023**, *223*, 109574. [\[CrossRef\]](#)
24. Consul, P.; Budhiraja, I.; Garg, D.; Kumar, N.; Singh, R.; Almogren, A.S. A Hybrid Task Offloading and Resource Allocation Approach for Digital Twin-Empowered UAV-Assisted MEC Network Using Federated Reinforcement Learning for Future Wireless Network. *IEEE Trans. Consum. Electron.* **2024**, *70*, 3120–3130. [\[CrossRef\]](#)
25. Zhang, P.; Su, Y.; Li, B.; Liu, L.; Wang, C.; Zhang, W.; Tan, L. Deep Reinforcement Learning Based Computation Offloading in UAV-Assisted Edge Computing. *Drones* **2023**, *7*, 213. [\[CrossRef\]](#)
26. Elgendy, I.A.; Meshoul, S.; Hammad, M. Joint task offloading, resource allocation, and load-balancing optimization in multi-UAV-aided MEC systems. *Appl. Sci.* **2023**, *13*, 2625. [\[CrossRef\]](#)
27. Seid, A.M.; Boateng, G.O.; Mareri, B.; Sun, G.; Jiang, W. Multi-Agent DRL for Task Offloading and Resource Allocation in Multi-UAV Enabled IoT Edge Network. *IEEE Trans. Netw. Serv. Manag.* **2021**, *18*, 4531–4547. [\[CrossRef\]](#)
28. Hao, J. Machine Learning for Road Active Safety in Vehicular Networks. Ph.D. Thesis, Institut Polytechnique de Paris, Palaiseau, France, 2024.
29. Lin, N.; Tang, H.; Zhao, L.; Wan, S.; Hawbani, A.; Guizani, M. A PDDQNLP algorithm for energy efficient computation offloading in UAV-assisted MEC. *IEEE Trans. Wirel. Commun.* **2023**, *22*, 8876–8890. [\[CrossRef\]](#)
30. Yan, J.; Zhao, X.; Li, Z. Deep Reinforcement Learning Based Computation Offloading in UAV-Assisted Vehicular Edge Computing Networks. *IEEE Internet Things J.* **2024**, *11*, 19882–19897. [\[CrossRef\]](#)

31. Bor-Yaliniz, R.I.; El-Keyi, A.; Yanikomeroglu, H. Efficient 3-D placement of an aerial base station in next generation cellular networks. In Proceedings of the 2016 IEEE International Conference on Communications (ICC), Kuala Lumpur, Malaysia, 22–27 May 2016; pp. 1–5. [[CrossRef](#)]
32. Shakya, A.K.; Pillai, G.; Chakrabarty, S. Reinforcement learning algorithms: A brief survey. *Expert Syst. Appl.* **2023**, *231*, 120495. [[CrossRef](#)]
33. Sutton, R.S. Learning to predict by the methods of temporal differences. *Mach. Learn.* **1988**, *3*, 9–44. [[CrossRef](#)]
34. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347.
35. Hinton, G.; Vinyals, O.; Dean, J. Distilling the knowledge in a neural network. *arXiv* **2015**, arXiv:1503.02531.
36. Gordon, G.; Tibshirani, R. Karush-kuhn-tucker conditions. *Optimization* **2012**, *10*, 725.
37. Al-Hourani, A.; Kandeepan, S.; Lardner, S. Optimal LAP Altitude for Maximum Coverage. *IEEE Wirel. Commun. Lett.* **2014**, *3*, 569–572. [[CrossRef](#)]
38. Sadia, R.; Akter, S.; Yoon, S. Ellipsoidal Trajectory Optimization for Minimizing Latency and Data Transmission Energy in UAV-Assisted MEC Using Deep Reinforcement Learning. *Appl. Sci.* **2023**, *13*, 12136 [[CrossRef](#)]
39. Tian, J.; Wang, D.; Zhang, H.; Wu, D. Service Satisfaction-Oriented Task Offloading and UAV Scheduling in UAV-Enabled MEC Networks. *IEEE Trans. Wirel. Commun.* **2023**, *22*, 8949–8964. [[CrossRef](#)]
40. Zheng, J.; Cai, Y.; Wu, Y.; Shen, X. Dynamic Computation Offloading for Mobile Cloud Computing: A Stochastic Game-Theoretic Approach. *IEEE Trans. Mob. Comput.* **2019**, *18*, 771–786. [[CrossRef](#)]
41. Akter, S.; Kim, D.Y.; Yoon, S. Task offloading in multi-access edge computing enabled UAV-aided emergency response operations. *IEEE Access* **2023**, *11*, 23167–23188. [[CrossRef](#)]
42. Romli, R.; Razali, A.F.; Ghazali, N.H.; Hanin, N.A.; Ibrahim, S.Z. Mobile augmented reality (AR) marker-based for indoor library navigation. *IOP Conf. Ser. Mater. Sci. Eng.* **2020**, *767*, 012062. [[CrossRef](#)]
43. Gherghina, A.; Olteanu, A.C.; Tapus, N. A marker-based augmented reality system for mobile devices. In Proceedings of the 2013 11th RoEduNet International Conference, Sinaia, Romania, 17–19 January 2013; pp. 1–6. [[CrossRef](#)]
44. Siriwardhana, Y.; Porambage, P.; Liyanage, M.; Ylianttila, M. A Survey on Mobile Augmented Reality with 5G Mobile Edge Computing: Architectures, Applications, and Technical Aspects. *IEEE Commun. Surv. Tutorials* **2021**, *23*, 1160–1192. [[CrossRef](#)]
45. Alhijawi, B.; Awajan, A. Genetic algorithms: Theory, genetic operators, solutions, and applications. *Evol. Intell.* **2024**, *17*, 1245–1256. [[CrossRef](#)]
46. Haarnoja, T.; Zhou, A.; Hartikainen, K.; Tucker, G.; Ha, S.; Tan, J.; Kumar, V.; Zhu, H.; Gupta, A.; Abbeel, P.; et al. Soft actor-critic algorithms and applications. *arXiv* **2018**, arXiv:1812.05905.

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