Learning and Batch-Processing Based Coded Computation with Mobility Awareness for Networked Airborne Computing

Baoqian Wang, *Student Member, IEEE*, Junfei Xie, *Senior Member, IEEE*, Kejie Lu, *Senior Member, IEEE*, Yan Wan, *Senior Member, IEEE*, and Shengli Fu, *Senior Member, IEEE*

***Abstract*—The implementation of many Unmanned Aerial Vehicle (UAV) applications (e.g., fire detection, surveillance, and package delivery) requires extensive computing resources to achieve reliable performance. Existing solutions that offload computation tasks to the ground may suffer from long communication delays. To address this issue, the Networked Airborne Computing (NAC) is a promising technique, which offers advanced onboard airborne computing capabilities by sharing resources among the UAVs via direct flight-to-flight links. However, NAC does not exist yet and enabling it requires overcoming many technical challenges, such as the high UAV mobility, and the uncertain, heterogeneous, and dynamic airspace. This paper addresses these challenges by 1) developing a Dynamic Batch-Processing based Coded Computation (D-BPCC) framework for achieving robust and adaptable cooperative airborne computing, and 2) designing deep reinforcement learning (DRL) based load allocation and UAV mobility control strategies for optimizing the system performance. As the first study to systematically investigate NAC, to the best of our knowledge, we evaluate the proposed methods through designing a NAC simulator and conducting comparative studies with four state-of-the-art distributed computing schemes. The results demonstrate the promising performance of the proposed methods.**

***Index Terms*—Networked airborne computing, unmanned aerial vehicle, coded distributed computing, reinforcement learning, load allocation, mobility control**

I. Introduction

Unmanned Aerial Vehicles (UAVs) have been widely used to assist different ground entities or networks, such as providing wireless services to cellular users, increasing connectivity in vehicular networks, delivering medical supplies to disaster areas, and offering computing services to ground users [1]–[3]. Among these applications, using UAVs to assist computing has recently drawn a growing interest. The implementation of advanced UAV functions (e.g., path planning, positioning, video processing, flight control) also requires a significant amount of computing resources. However, due to small payload, the computing capability of most existing UAV platforms is very limited. To execute computation-intensive tasks, the existing solutions are to offload these tasks to ground servers or remote clouds, which can, however, incur significant delays or even failures [4].

Baoqian Wang is with the Department of Electrical and Computer Engineering, University of California, San Diego, and San Diego State University, San Diego, CA, 92182 (e-mail: bawang@ucsd.edu).

Junfei Xie is with the Department of Electrical and Computer Engineering, San Diego State University, San Diego, CA, 92182 (e-mail: jxie4@sdsu.edu). Corresponding author.

Kejie Lu is with the Department of Computer Science and Engineering, University of Puerto Rico at Mayagüez, Mayagüez, Puerto Rico, 00681, e-mail: (kejie.lu@upr.edu).

Yan Wan is with the Department of Electrical Engineering, University of Texas at Arlington, Arlington, Texas, 76019 (e-mail: yan.wan@uta.edu).

Shengli Fu is with the Department of Electrical Engineering, University of North Texas, Denton, Texas, 76201 (e-mail: Shengli.Fu@unt.edu).

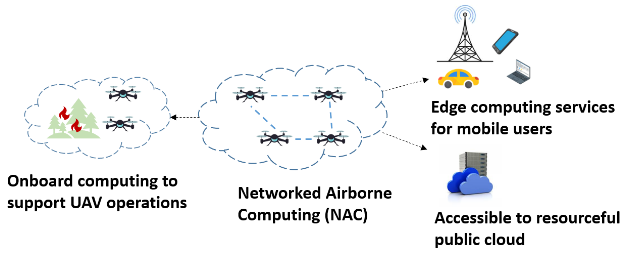

Edge computing services for mobile users
Onboard computing to support UAV operations
Networked Airborne Computing (NAC)
Accessible to resourceful public cloud

Fig. 1: Illustration of Networked Airborne Computing (NAC) paradigm.

To address the aforementioned issues, a promising technique is the Networked Airborne Computing (NAC) [5] that can offer advanced onboard airborne computing capabilities. NAC is a new computing paradigm formed by aerial vehicles connected with direct flight-to-flight communication links (see Fig. 1). The advantages of UAV-based NAC include low latency, transportability, infrastructure-free, unmanned maneuvering, fast deployment, wide-coverage, and low cost. It can not only enhance UAVs' system performance by allowing advanced algorithms to be implemented onboard of UAVs and hence benefit a wide range of existing UAV applications, but also give rise to a variety of new applications. For example, it can facilitate data collection, processing, and distribution for Internet of Things (IoT) devices, and can function as Mobile Edge Computing (MEC) servers [6] to provide computing services for ground users, etc.

Despite the exciting advantages and broad applications, enabling a UAV-based NAC requires overcoming many technical challenges. For example, when UAVs operate in the complex and uncertain airspace with high mobility, the fast node movement, line-of-sight effect, and node leaving and joining can cause frequent topology changes, link failures, data losses, and task interruptions. Moreover, the various uncertainties (e.g., winds and other vehicles) present in the airspace can disturb the communication among the UAVs, bringing additional challenges for robust computing. Currently, the research on UAV-based NAC is still in its early stage, and most existing studies have been focused on the UAV-assisted MEC [7]–[14] with a single UAV, which is just one of many possible applications of NAC. Specifically, these studies investigate how to provide the best computing services to ground users, via properly allocating resources and planning UAV trajectories. Moreover, in these studies, the UAVs act alone without collaboration and follow trajectories that are pre-planned. The locations of ground users are assumed to be known and fixed.

In this paper, we directly tackle the key technical challenges of NAC to achieve robust cooperative airborne computing onboard of multiple networked UAVs. In our study, we consider that a NAC system can be formed in different ways, and there are two realistic formation scenarios. First, the NAC system is formed by UAVs operated by different owners in an opportunistic manner, e.g., when cargo drones owned by different companies are serving the same area. Here, the mobility of the UAVs is uncontrollable, unknown, and can be considered random. Second, the NAC system is formed by UAVs operated by the same owner, e.g., in multi-UAV applications like multi-UAV surveillance, search and rescue. In this scenario, the mobility of the UAVs can be controlled and proactively planned by the owner to facilitate computing. In this study, we consider both formation scenarios and develop innovative computation schemes for the two scenarios to enable efficient, robust, and adaptable cooperative airborne computing in an uncertain, heterogeneous, and dynamic airspace. The main contributions are summarized as follows:

1) *Dynamic batch-processing based coded computation (D-BPCC) framework:* This framework features a dynamic batch-processing based procedure and applies the coding theory to address the uncertainties phenomenal in a dynamic NAC system and to improve the efficiency, robustness, and adaptability of the computing system.
2) *Deep reinforcement learning (DRL) based optimization and control:* DRL-based online decision-making strategies are designed to optimize the system performance and control UAV mobility. Compared to the conventional numerical optimization methods [7]–[14], DRL-based strategies do not require any knowledge of the communication, computation, or UAV mobility models, can be quickly deployed in any NAC systems, and generate solutions in real time.
3) *Two typical NAC formation scenarios:* We address two typical NAC formation scenarios by applying the DRL-based and D-BPCC-based schemes. To the best of our knowledge, these two scenarios have not been investigated in the literature.
4) *Comprehensive simulation studies:* We conduct comprehensive simulation studies to evaluate the performance of the proposed methods for the two NAC formation scenarios from several aspects. We also implement four state-of-the-art distributed computing schemes as the benchmarks for comparison studies.

The rest of the paper is organized as follows. Section II presents the related work. Section III describes the NAC system and the computation tasks performed on it. In Section IV, the D-BPCC framework is introduced to enable the NAC system to execute computation tasks robustly in a dynamic and uncertain airspace. To optimize the system performance, two optimization problems are then formulated in the same section for the two NAC formation scenarios. Sections V and VI introduce the DRL-based methods to solve the two optimization problems. The performance of the proposed methods is evaluated via simulations and comparative studies in Section VII. Section VIII concludes the paper.

II. Related Work

In this section, we review related work in four areas: networked airborne computing (NAC), UAV-assisted mobile edge computing (MEC), DRL-based UAV-assisted networks, and coded distributed computing.

A. Networked Airborne Computing

Most existing works on UAV-assisted computing focus on MEC [7]–[14], where UAVs function as servers to provide computing services to ground users. Studies that explore resource sharing among UAVs in uncertain airspace via direct flight-to-flight links are very limited. In [5], the concept of NAC, its advantages and design guidelines were introduced. In [15], [16], we investigated the hardware and software design for a prototype of the NAC platform. In [17], we considered a NAC system formed by static UAVs hovering in the air and developed a coded distributed computing scheme to achieve robust computation of matrix multiplication tasks.

A more general concept, called mobile ad hoc computing or mobile ad hoc cloud [18]–[20], was coined recently in [18], which refers to any computing systems formed by mobile devices with resources shared among each other. Existing studies have mostly considered smartphones [21]–[23] or ground vehicles [24] as the main computing resource providers. Nevertheless, substantial differences exist between NAC and ground-based mobile ad hoc computing due to the unique features of UAVs such as high 3-D mobility, highly uncertain operating environment with significant impacts on aerial dynamics, stringent safety requirements, mechanical and aerospace constraints. Existing solutions cannot address these new technical challenges of NAC.

B. UAV-assisted Mobile Edge Computing

MEC addresses the high transmission latency of remote cloud-based computing paradigms by deploying cloud resources at the network edge close to users [6]. In UAV-assisted MEC [25], UAV with computing power functions as an edge server to provide computing services for ground users. To improve the quality of service (QoS), joint computation offloading and UAV trajectory designs were investigated in [7]–[14], where the UAV follows an optimized trajectory for serving multiple static ground users. Recently, a few studies extended the problem to multiple UAVs [26]–[28]. In particular, [26], [27] made UAVs hover statically over ground users and investigated the optimal placement of UAVs and task assignment. In [28], the trajectories of UAVs were optimally designed, while jointly considering the optimal bit allocation and task assignment. In [29], the authors considered an MEC system formed by stationary and UAV-based quasi-stationary MEC servers, and investigated how to form coalitions with shared resources for MEC servers to better serve their users.

To solve the aforementioned optimization problems, most studies [7]–[14], [26]–[28] assumed known and time-invariant communication and computation models and attempted to derive exact solutions. However, this assumption often does not hold in reality.

C. DRL-based UAV-Assisted Networks

To manage unknown and complex UAV-assisted networks, DRL has been explored in many recent studies [30]–[36]. For example, multi-agent reinforcement learning (MARL) was used in [36] to optimize the wireless energy transfer between UAVs and flying energy resources. Deep Q Network (DQN) was employed in [35] to achieve efficient dispatch of UAVs as relays in vehicular networks. In [30], DQN was used for UAV path planning in UAV-assisted MEC to minimize energy consumption and maximize task completion efficiency. In [31], MARL was applied to plan UAV trajectories, based on which an optimization approach was developed for computation offloading. DRL has also been explored to generate offloading decisions in UAV-assisted MEC [33], [34], but it hasn't been applied to NAC.

D. Coded Distributed Computing

The resilience of distributed computing systems to uncertain system disturbances can be enhanced using the coded computation techniques. The key idea is to apply the coding theory to generate redundant computations for reducing the impact of disturbances. This idea has been explored for different computation problems, such as matrix multiplications [37]–[42], linear inverse problems [43], convolution [44], deep neural networks [45], map-reduce [46], and MARL [47].

Nevertheless, most of these approaches were developed based on multiple assumptions (e.g., homogeneous and static computing nodes, known and time-invariant data transfer behavior and computing power) and thus, cannot be directly applied for NAC.

Recently, coded distributed computing (CDC) has also been explored to facilitate MEC. For example, an error-correcting-code-inspired strategy was proposed in [48] to execute computation tasks in edge servers. Paper [49] introduced a coding scheme that combines a rateless code for improving system resiliency and an irregular-repetition code for reducing the communication latency. However, both approaches assume homogeneous and static computing nodes. In [50], a Lagrange coded computing-based framework was developed to enable fast and secure computation in MEC with heterogeneous but static edge servers. Papers [39], [40] are closely related to this work, which consider a MEC system with heterogeneous and mobile computing nodes. To reduce task completion delay, a coded computation framework called the coded cooperative computation protocol (C3P) was developed. Although this framework can address the first NAC formation scenario with uncontrollable UAVs, it cannot address the second scenario that requires UAV mobility control. Moreover, as we will show in the Simulation Studies section (Sec. VII), C3P is vulnerable to frequent network changes caused by node movement.

Table I compares the studies that are closely related to this work from various aspects including application, objective, offloading direction, method, planning horizon, mobility control, computation load optimization, and straggler mitigation.

III. NAC SYSTEM

In this paper, we consider a NAC system that consists of multiple UAVs flying at the same altitude. The system can be either formed by UAVs with random mobility, i.e., uncontrollable and unpredictable, or UAVs that can be proactively maneuvered. The onboard computation tasks to be executed by the UAVs cooperatively are assumed to be matrix-vector multiplications $\mathbf{A}\mathbf{x}$, where $\mathbf{A} \in \mathbb{R}^{p\times m}$ is a pre-stored matrix. Matrix-vector multiplication is considered here as it is the building block for many computation tasks, especially machine learning based applications such as collaborative filtering recommender systems [51] and object detection [52]. The proposed computation framework can be easily extended for other problems, such as convolution [53] and distributed path planning [54].

The UAV that receives a sequence of input vectors, $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_K\}$, to process is referred to as the *master node*, where $\mathbf{x}_j \in \mathbb{R}^{m\times 1}$, $j \in [K] := \{1, 2, \ldots, K\}$ and K is the total number of input vectors. Due to limited computing power, executing each task in a single UAV can be time consuming when the task size is large. To speed up the computation, the master node cooperates with its neighboring N UAVs within the communication range by sharing with them the computation loads. These neighboring UAVs, referred to as the *worker nodes*, execute tasks assigned by the master node and send back the obtained results. The master node then aggregates the results to output the final values.

This paper seeks the fastest way for the aforementioned NAC system to complete all tasks. The desired features of the approach include: 1) Resilience to the various uncertain system disturbances prominent in the UAV network, such as communication bottlenecks in the network traffic, link/node failures, package losses, and slow-downs of computing nodes; 2) Adaptivity for unpredictable network changes such as random node movement, topology and resource (e.g., communication, computing, and energy) changes; and 3) quick deployment without requiring any knowledge of the system models.

IV. DYNAMIC BATCH-PROCESSING BASED CODED COMPUTATION FRAMEWORK

In this section, we first introduce a dynamic batch-processing based coded computation (D-BPCC) framework to enable robust cooperative airborne computing in an uncertain airspace. Under this framework, we then formulate the system optimization problems mathematically for the two different NAC formation scenarios.

Tab. I: A Comparison of existing studies on UAV or mobile device assisted computing

Ref.	Application	Objective	Offloading Direction	Method	Plan Horizon	Mobility Control	Comput. Load Optim.	Straggler Mitig.
[7]	UAV-assisted MEC	Efficiently offload partial computation tasks from ground users to the UAV.	Ground to UAV	Numerical	Offline	✓	✓	✗
[8]	UAV-assisted MEC	Use UAVs to provide computing services and transmit wireless power to mobile users	Ground to UAV	Numerical	Online	✓	✓	✗
[9]	UAV-assisted MEC	Offload computations from ground IoT nodes to UAVs	Ground to UAV	DRL	Online	✗	✓	✗
[10]	UAV-assisted MEC	Minimize mobile users' energy consumption while offloading mobile applications to run on the UAV	Ground to UAV	Numerical	Online	✓	✓	✗
[11]	UAV-assisted MEC	Maximize computation bits of UAV-assisted MEC while satisfying energy constraints	Gound to UAV	Numerical	Offline	✓	✓	✗
[12]	UAV-assisted MEC	Maximize energy efficiency for UAV-assisted MEC	Ground to UAV	Numerical	Offline	✓	✓	✗
[13]	UAV-assisted MEC	Minimize weighted sum of energy consumption of UAV and users	Ground to UAV	Numerical	Offline	✓	✓	✗
[14]	UAV-assisted MEC	Minimize energy consumption for task offloading between IoT devices and the UAV	Ground to UAV	Numerical	Offline	✓	✓	✗
[26], [27]	UAV-assisted MEC	Minimize system energy consumption by jointly optimizing task scheduling and UAV deployment	Ground to UAV	Numerical	Offline	✗	✓	✗
[28]	UAV-assisted MEC	Minimize total energy consumption by jointly optimizing task scheduling, UAVs' trajectories and bit allocation	Ground to UAV	Numerical	Online	✓	✓	✗
[29]	UAV-assisted MEC	Efficiently offload computations from mobile edge severs to UAVs	Ground to UAV	DRL	Online	✗	✓	✗
[30]	UAV-assisted MEC	Ensure QoS of MEC system while satisfying energy constraints	Ground to UAV	DRL	Online	✓	✗	✗
[31]	UAV-assisted MEC	Jointly minimize energy consumption of users and maximize service fairness of UAVs	Ground to UAV	DRL	Online	✓	✓	✗
[39], [40]	MEC	Minimize task completion time through cooperative coded computation on mobile edge devices	Mobile device to mobile device	Numerical	Online	✗	✓	✓
This Work	UAV-based NAC	Jointly minimize computation time and mission flight time	UAV to UAV	DRL	Online	✓	✓	✓

A. D-BPCC Framework

The D-BPCC framework (see Fig. 2) exploits the coding theory to enhance system resilience to uncertain system disturbances and uses a dynamic batch-processing based procedure (extended from our previous design for static networks [42]) to make the NAC system adaptable to unpredictable network changes. In particular, in each worker node $i \in [N] := \{1, 2, \ldots, N\}$, we encode matrix $\mathbf{A}$ into a new matrix $\hat{\mathbf{A}}_i \in \mathbb{R}^{p\times m}$ using the following equation

$$\hat{\mathbf{A}}_i = \mathbf{G}_i\mathbf{A},$$

where $\mathbf{G}_i \in \mathbb{R}^{p\times p}$ is an encoding matrix satisfying the condition that any p rows of the concatenated encoding matrix $\mathbf{G} = [\mathbf{G}_1; \mathbf{G}_2; \ldots; \mathbf{G}_N] \in \mathbb{R}^{Np\times p}$ are linearly independent. This step is computed offline, and $\hat{\mathbf{A}}_i$ and $\mathbf{G}$ are pre-stored in each worker node i, assuming the storage space of each worker node is sufficiently large.

Once receiving an input vector $\mathbf{x}_j$, the master node sends $\mathbf{x}_j$ to each worker node i. At the same time, it also notifies each node i the number of rows $h_{i,j}$ of $\hat{\mathbf{A}}_i$ to be processed at a time. In another word, each worker node i will evenly divide $\hat{\mathbf{A}}_i$ row-wise into $\lceil \frac{p}{h_{i,j}} \rceil$ submatrices as $[\hat{\mathbf{A}}_{i,1}, \ldots, \hat{\mathbf{A}}_{i,\lceil \frac{p}{h_{i,j}} \rceil}]$, where each submatrix has $h_{i,j}$ rows except the last one $\hat{\mathbf{A}}_{i,\lceil \frac{p}{h_{i,j}} \rceil}$ that has $p - (\lceil \frac{p}{h_{i,j}} \rceil - 1)h_{i,j}$ rows. Each submatrix will then be multiplied with $\mathbf{x}_j$ one by one, i.e., $\hat{A}_{i,k}\mathbf{x}_j, \forall k \in [\lceil \frac{p}{h_{i,j}} \rceil]$. For ease of reference, we hereinafter call each submatrix of $\hat{\mathbf{A}}_i$ as a *batch*, and $h_{i,j}$ as the *batch size*. Once a batch is processed, the worker node will send the result back to the master node immediately and move on to process the next batch.

The worker nodes will stop processing after receiving the notification from the master node, who will send such notification after it receives sufficient results for generating

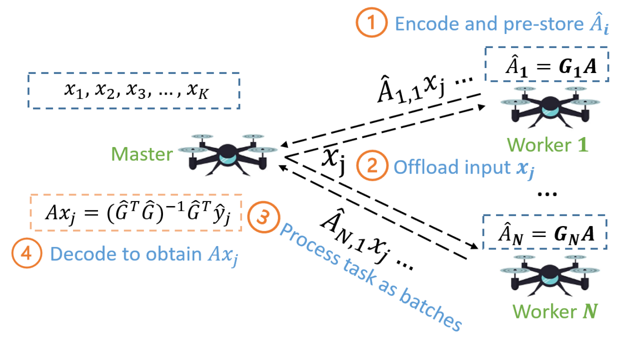

1 Encode and pre-store $\hat{A}_i$
$x_1, x_2, x_3, \ldots, x_K$
$\hat{A}_1 = G_1 A$
$\hat{A}_{1,1}x_j \cdots$
Worker 1
Master
x_j
2 Offload input x_j
$Ax_j = (\hat{G}^T\hat{G})^{-1}\hat{G}^T\hat{y}_j$
3 Process task as batches
$\hat{A}_{N,1}x_j \cdots$
$\hat{A}_N = G_N A$
4 Decode to obtain Ax_j
Worker N

Fig. 2: Cooperative airborne computing of matrix-vector multiplication tasks under the D-BPCC framework.

the output. In particular, let $\hat{\mathbf{y}}_j$ denote the results received at the master node by a certain time, which can be represented by

$$\hat{\mathbf{y}}_j = \hat{\mathbf{G}}\mathbf{A}\mathbf{x}_j,$$

where $\hat{\mathbf{G}}$ is a submatrix of $[\mathbf{G}_1; \mathbf{G}_2; \ldots; \mathbf{G}_N]$. Then, the master node can generate the output using the following equation

$$\mathbf{A}\mathbf{x}_j = (\hat{\mathbf{G}}^T\hat{\mathbf{G}})^{-1}\hat{\mathbf{G}}^T\hat{\mathbf{y}}_j,$$

as long as the length of $\hat{\mathbf{y}}_j$ is larger than or equal to p, i.e., $|\hat{\mathbf{y}}_j| \geq p$.

The following lemma shows the resilience of D-BPCC to uncertain stragglers.

Lemma 1. *For a distributed computing system with a master node and N worker nodes, D-BPCC can tolerate up to $N-1$ worker nodes failures when processing matrix-vector multiplication tasks.*

Proof. As each worker node i pre-stores the encoded matrix $\hat{\mathbf{A}}_i = \mathbf{G}_i\mathbf{A}$ with $\hat{\mathbf{A}}_i \in \mathbb{R}^{p\times m}$, given an input $\mathbf{x} \in \mathbb{R}^{m\times 1}$, the aggregated results computed by the node are sufficient for obtaining the value of $\mathbf{A}\mathbf{x}$ by $\mathbf{A}\mathbf{x} = \mathbf{G}_i^{-1}\hat{\mathbf{A}}_i\mathbf{x}$. Therefore, when there are $N-1$ or fewer worker nodes failing to return results, the master node can utilize the results from other nodes to recover the final value. □

Remark 1. *According to Lemma 1, with D-BPCC, the master node can complete each computation task $\mathbf{A}\mathbf{x}_j$ as long as there is a functioning worker node. If the master node also shares certain workload (i.e., also being one of the worker nodes), any network changes or uncertain system disturbances won't cause tasks to fail as long as the master node is functional.*

Remark 2. *In the special case that $N-1$ worker nodes fail, the remaining functional worker node i will compute the whole task $\hat{\mathbf{A}}_i\mathbf{x}$. Although this may lead to more computation time compared with directly performing the task at the master node without distributed computing, the probability of this happening is usually small especially when N is large.*

The high resilience makes D-BPCC suitable for NAC in an uncertain and dynamic airspace. Processing tasks as small batches also naturally handles node heterogeneity as nodes with more computing or communication resources will process more batches. Moreover, with batch processing, the master node will continuously receive partial results from the worker nodes, which can be utilized to generate approximated outputs. This feature is crucial for safe UAV operations that require quick responses to environmental changes such as wind, birds, obstacles, and other UAVs.

B. Problem Formulation

In D-BPCC, the batch size $h_{i,j}$ is a key control variable to be determined, which will impact the system performance. In particular, when $h_{i,j}$ is large (e.g., equal to p), very few worker nodes will essentially contribute to the computation and their resources are hence underutilized. On the contrary, when $h_{i,j}$ is small (e.g., equal to 1), the frequent data transmissions by the worker nodes may generate large overhead and communication traffic. Another key factor that will impact the computing performance is the relative distance between two UAVs, which affects the data transmission time. In the NAC formation scenario where UAVs move randomly with uncontrollable mobility, we exploit the optimization of the batch size $h_{i,j}$ to minimize the impact of random UAV mobility and other uncertain system disturbances, and make the system adaptable to uncertain network changes. In the scenario where UAVs are controllable, we exploit the benefit of UAV mobility control to computing and jointly optimize the batch size and UAV mobility.

To formulate the system optimization problems mathematically, we first introduce the evaluation metrics for the system performance. Denote the time required by each worker node i to process $b_{i,j}$ batches for the j-th task as $T_{i,j}$. Then $T_{i,j}$ can be captured by the following equation.

$$T_{i,j} = T_{i,j}^{comm} + T_{i,j}^{comp}, \quad (1)$$

which includes the communication time $T_{i,j}^{comm}$ and the computation time $T_{i,j}^{comp}$. The communication time $T_{i,j}^{comm}$ can be captured by

$$T_{i,j}^{comm} = T_{i,j}^{comm}(\mathbf{x}_j) + T_{i,j}^{comm}(\hat{A}_{i,b_{i,j}}\mathbf{x}_j)$$

where $T_i^{comm}(\mathbf{x}_j)$ is the time spent to send the input vector $\mathbf{x}_j$ from the master node to the worker node i and $T_{i,j}^{comm}(\hat{A}_{i,b_{i,j}}\mathbf{x}_j)$ is the time spent to send the last batch computation result from the worker node i back to the master node. Note that there is no break between two batches. Once a worker node completes a batch, it will immediately move on to process the next batch and at the same time transmit the computation result of the previous batch to the master node.

The computation time $T_{i,j}^{comp}$ in (1) can be captured by

$$T_{i,j}^{comp} = \sum_{k=1}^{b_{i,j}} T_{i,j}^{comp}(\hat{A}_{i,k}, \mathbf{x}_j)$$

where $T_{i,j}^{comp}(\hat{\mathbf{A}}_{i,k}, \mathbf{x}_j)$ is the time spent by the worker node i to compute one batch $\hat{A}_{i,k}\mathbf{x}_j$. Note that the communication and computation times are affected by various factors and finding perfect models for them is very challenging considering the uncertain and dynamic airspace. In contrast with most existing studies that assume the existence of perfect communication and computation models, we do not make such assumptions in this study.

With $T_{i,j}$, the completion time of each task $j \in [K]$ can then be represented by:

$$T_j = \min_t \{t | R_j(t) \geq p\}$$

where $R_j(t) = \sum_{i=1}^{N} b_{i,j} h_{i,j} \mathbb{1}_{T_{i,j} \leq t}$ is the total number of rows of inner product results for task j that the master node has received by time t. $\mathbb{1}$ is the indicator function [55].

The mathematical formulations of the optimization problems for the two NAC formation scenarios are described as follows.

1) Problem 1: Consider the scenario where the mobility of UAVs is random and cannot be controlled. Assuming that the positions and velocities of the UAVs can be observed through sensing and estimation, we aim to minimize the total task completion time by optimizing the batch size $h_{i,j}$, which is formulated as

$$\begin{aligned}
\mathcal{P}_1 : \min_{\substack{h_{i,j} \\ \forall i \in [N], \forall j \in [K]}} \quad & J_1 = \sum_{j=1}^{K} T_j \\
\text{s.t.} \quad & \mathcal{C}_1 : h_{i,j} \in \mathbb{Z}^+, \forall i \in [N], \forall j \in [K] \\
& \mathcal{C}_2 : h_{i,j} \leq p, \forall i \in [N], \forall j \in [K] \\
& \mathcal{C}_3 : \mathbf{p}_{i,t+\Delta T} = f_i(\mathbf{p}_{i,t}, \mathbf{v}_{i,t}, \Delta T), \\
& \qquad\qquad \forall i \in [N+1] \\
& \mathcal{C}_4 : T_j = \min_t \{t | R_j(t) \geq p\}, \forall j \in [K] \\
& \mathcal{C}_5 : R_j(t) = \sum_{i=1}^{N} b_{i,j} h_{i,j} \mathbb{1}_{T_{i,j} \leq t}, \forall j \in [K]
\end{aligned} \tag{2}$$

In constraint $\mathcal{C}_3$, $\mathbf{p}_{i,t}, \mathbf{v}_{i,t}$ denote the position and velocity of UAV i at time t, respectively, where $i \in [N+1]$ with the master node indexed by $N+1$. $f_i(\cdot)$ describes the movement behavior of each worker node i, whose specific formula is unknown, but the velocity of each UAV is assumed to be fixed over a small time period ΔT.

2) Problem 2: Consider the scenario where the mobility of UAVs is controllable, and each UAV has a target location to reach while performing cooperative airborne computing. The goal is to simultaneously minimize the total task completion time and the total UAV flight time. To achieve this goal, we formulate the following optimization problem that jointly optimizes UAVs' velocities and batch sizes.

$$\begin{aligned}
\mathcal{P}_2 : \min_{\substack{h_{i,j}, \mathbf{v}_{i,j} \\ \forall i \in [N], \forall j \in [K]}} \quad & J_2 = \omega \sum_{j=1}^{K} T_j + \sum_{i=1}^{N+1} T_i^{flight} \\
\text{s.t.} \quad & \mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3, \mathcal{C}_4, \mathcal{C}_5 \\
& \mathcal{C}_6 : \mathbf{v}_{i,t} = \mathbf{v}_{i,j}, t_j \leq t < t_{j+1}, \forall i \in [N+1] \\
& \mathcal{C}_7 : \mathbf{p}_{i,T_i^{flight}} = \mathbf{g}_i, \forall i \in [N+1] \\
& \mathcal{C}_8 : |\mathbf{p}_{i,t} - \mathbf{p}_{j,t}| \leq \epsilon, \forall i, j \in [N+1], i \neq j \\
& \mathcal{C}_9 : \mathbf{v}_{min} \leq \mathbf{v}_{i,j} \leq \mathbf{v}_{max}, \forall i \in [N+1], j \in [K]
\end{aligned} \tag{3}$$

where T_i^{flight} is the flight time of UAV i and ω is a weight that trades off between total task completion time and total flight time. When executing each task j, we let each UAV i fly at a velocity of $\mathbf{v}_{i,j}$, which remains unchanged during the task execution period $[t_j, t_{j+1})$, where t_j is the start time of task j and $t_{j+1} = t_j + T_j$. Constraint $\mathcal{C}_7$ ensures that each UAV will reach its target location specified by $\mathbf{g}_i$. Constraint $\mathcal{C}_8$ prevents collisions among the UAVs, where ϵ is the minimum safety distance between two UAVs.

V. DRL-based Solution to $\mathcal{P}_1$

In this section, we solve problem $\mathcal{P}_1$ in (2) by exploiting DRL, specifically, the Deep Deterministic Policy Gradient (DDPG) method [56], which does not need any knowledge of the communication, computation or UAV mobility models, and can be quickly deployed in any NAC systems to make decisions online in real time. We first convert the optimization problem into a reinforcement learning (RL) problem, and then describe the solution to the resulting problem.

A. RL based Formulation for $\mathcal{P}_1$

To convert $\mathcal{P}_1$ into a RL problem, we first model this problem as a Markov Decision Process (MDP) characterized by a quintuple $(\mathbf{s}_t, \mathbf{a}_t, \zeta, r, \mu)$ with each component described as follows.

1) State $\mathbf{s}_t$: The system state at time t includes distances $d_{i,t}$ between each worker node and the master node, and velocities $\mathbf{v}_{i,t}$ of all nodes. Specifically,

$$\mathbf{s}_t = [d_{1,t}, d_{2,t}, \ldots, d_{N,t}, \mathbf{v}_{1,t}, \mathbf{v}_{2,t}, \ldots, \mathbf{v}_{N+1,t}]^\top \in \mathcal{S},$$

where $\mathcal{S}$ denotes the state space.

2) Action $\mathbf{a}_t$: In $\mathcal{P}_1$, the batch size $h_{i,j}$ for each worker node i and task j is the control variable to be determined when executing the task j, hence the action to take. The action taken at time t_j (start time of task j) is then defined as follows

$$\mathbf{a}_{t_j} = [h_{1,j}, h_{2,j}, h_{3,j}, \ldots, h_{N,j}]^\top \in \mathcal{A}$$

where $\mathcal{A}$ is the action space. Note that each $h_{i,j}$ in $\mathbf{a}_{t_j}$ should satisfy constraints $\mathcal{C}_1$ and $\mathcal{C}_2$ in $\mathcal{P}_1$, i.e., $h_{i,j} \in \mathbb{Z}^+$ and $h_{i,j} \leq p$.

3) Transition function ζ: The transition function describes the transition from the current state to the next state given the current action, and outputs the probability distribution over the next state, i.e., $\zeta : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$. In $\mathcal{P}_1$, given the state $\mathbf{s}_{t_j}$ and action $\mathbf{a}_{t_j}$ at time t_j, the next state of interest is the state $\mathbf{s}_{t_{j+1}}$ at the start time t_{j+1} of the next task $j+1$, as t_{j+1} is the time to make the next decision. To obtain $\mathbf{s}_{t_{j+1}}$, the computing nodes take action $\mathbf{a}_{t_j}$ and execute task j. The next state $\mathbf{s}_{t_{j+1}}$ can then be obtained by observing the positions and velocities of all nodes at time $t_{j+1} = t_j + T_j$. Note that in our settings, the explicit form of the transition function ζ is unknown.

4) Reward function r: As the goal of $\mathcal{P}_1$ is to minimize the total task completion time, we define the reward function as follows:

$$r(\mathbf{s}_{t_j}, \mathbf{a}_{t_j}) = -\omega_1 T_j$$

which can be obtained based on the state $\mathbf{s}_{t_j}$ and action $\mathbf{a}_{t_j}$. A larger reward indicates less time taken for computing a task j.

5) Policy function μ: The policy function determines the action to take given the current state, which can be deterministic or stochastic. We here consider a deterministic policy function that maps the state space to the action space, i.e., $\mu : \mathcal{S} \to \mathcal{A}$ and $\mathbf{a} = \mu(\mathbf{s})$. In $\mathcal{P}_1$, the policy function is called only at the start time t_j of each task j.

With the above MDP setting, we can then convert $\mathcal{P}_1$ into a RL problem that determines the optimal policy $\mu^*(\mathbf{s})$ such that the following expected cumulative discounted reward is maximized,

$$Q^\mu(\mathbf{s}, \mathbf{a}) := \mathbb{E}_{\substack{\mathbf{s}_{t_j} \sim \zeta \\ \mathbf{a}_{t_j} = \mu(\mathbf{s}_{t_j})}} \left[\sum_{j=1}^{K} \gamma^{j-1} r(\mathbf{s}_{t_j}, \mathbf{a}_{t_j}) | \mathbf{s}_{t_1} = \mathbf{s}, \mathbf{a}_{t_1} = \mathbf{a} \right]$$

where $\mathbf{s} \in \mathcal{S}$, $\mathbf{a} \in \mathcal{A}$, and $\gamma \in (0, 1]$ is a discount factor. $Q^\mu(\mathbf{s}, \mathbf{a})$ is also known as the action value function or Q function. The optimal policy can be obtained by

$$\mu^*(\mathbf{s}) \in \arg\max_{\mathbf{a}} Q^*(\mathbf{s}, \mathbf{a})$$

where $Q^*(\mathbf{s}, \mathbf{a})$ is the optimal Q function obtained by $Q^*(\mathbf{s}, \mathbf{a}) := \max_\mu Q^\mu(\mathbf{s}, \mathbf{a})$.

B. Deterministic Policy Gradient Method

To solve the above RL problem, the key is to derive the optimal Q function $Q^*(\mathbf{s}, \mathbf{a})$, which can be obtained by using the Bellman equation [57] as follows

$$Q^*(\mathbf{s}, \mathbf{a}) = \mathbb{E}_{\mathbf{s}' \sim \zeta} \left[r(\mathbf{s}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} Q^*(\mathbf{s}', \mathbf{a}') \right] \tag{4}$$

where $\mathbf{s}' \sim \zeta(\mathbf{s}, \mathbf{a})$ denotes the next state. As the state space is continuous, $Q^*(\mathbf{s}, \mathbf{a})$ cannot be directly computed. Instead, we approximate $Q^*(\mathbf{s}, \mathbf{a})$ using a parameterized non-linear function, denoted as $Q(\mathbf{s}, \mathbf{a}; \theta)$, where θ is the parameter. We delay the design of the non-linear function to the next subsection.

It is noted that in (4), the transition function ζ is needed for computing the optimal Q function, whose explicit form is, however, unknown. To address this challenge, we introduce a learning *agent* to interact with the NAC system, which is the *environment*, and collect transition data to approximate the transition function ζ. The transition data to be collected includes the state, action, reward, and the next state, i.e., $(\mathbf{s}, \mathbf{a}, r, \mathbf{s}')$, and are stored in a *replay buffer* denoted by $\mathcal{D}$. Equation (4) can then be rewritten as

$$Q(\mathbf{s}, \mathbf{a}; \theta) \approx \mathbb{E}_{(\mathbf{s}, \mathbf{a}, r, \mathbf{s}') \sim \mathcal{D}} \left[r(\mathbf{s}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} Q(\mathbf{s}', \mathbf{a}'; \theta) \right],$$

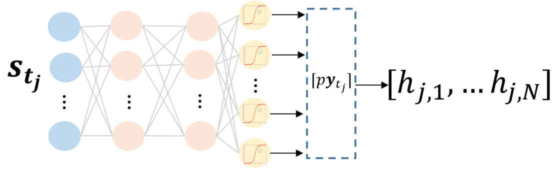


Fig. 3: DNN representation of the policy functions for computation load optimization.

with $Q^*(\mathbf{s}, \mathbf{a})$ approximated by $Q(\mathbf{s}, \mathbf{a}; \theta)$. This equation can be solved to obtain the optimal Q function by minimizing the following error,

$$J(\theta) = \mathbb{E}_{(\mathbf{s}, \mathbf{a}, r, \mathbf{s}') \sim \mathcal{D}} \left[\left(Q(\mathbf{s}, \mathbf{a}; \theta) - \left(r + \gamma \max_{\mathbf{a}'} Q(\mathbf{s}', \mathbf{a}'; \theta) \right) \right)^2 \right].$$

However, we note that directly computing $\max_{\mathbf{a}'} Q(\mathbf{s}', \mathbf{a}'; \theta)$ in the above error function is difficult considering the large state and action spaces. To address this issue, we introduce a policy approximator denoted by $\mu(\mathbf{s}; \phi)$ with parameter ϕ to output the action using $\mathbf{a} = \mu(\mathbf{s}; \phi)$. We then learn the parameters of the policy and Q functions to minimize the error function. Moreover, to stabilize the training procedure, we introduce a target policy function μ' and a target Q function Q' with the same function representations and initial weights as the original policy and Q functions. The error function finally used for finding the optimal Q function is then given as

$$J(\theta) = \mathbb{E}_{(\mathbf{s}, \mathbf{a}, r, \mathbf{s}') \sim \mathcal{D}} \left[(Q(\mathbf{s}, \mathbf{a}; \theta) - (r + \gamma Q'(\mathbf{s}', \mu'(\mathbf{s}'; \phi); \theta)))^2 \right]$$

where the parameters ϕ of the policy function is updated using the policy gradient theorem by

$$\nabla_\phi J(\phi) = \mathbb{E}_{(\mathbf{s}, \mathbf{a}, r, \mathbf{s}') \sim \mathcal{D}} \left[\nabla_\phi \mu(\mathbf{s}; \phi) \nabla_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a}; \theta) \right]$$

C. Deep Neural Network based Function Representation

To approximate the policy functions, $\mu(\mathbf{s})$ and $\mu'(\mathbf{s})$, and the Q functions, $Q(\mathbf{s}, \mathbf{a})$ and $Q'(\mathbf{s}, \mathbf{a})$, we adopt deep neural networks (DNNs) considering their powerful approximation capability. For the policy functions, we design a DNN with four fully connected layers. The input layer consists of $3N+2$ units representing the system state at time t_j (start time of task $j \in [K]$), denoted by $\mathbf{s}_{t_j}$. The output layer consists of $N+1$ sigmoid units [58], denoted by $\mathbf{y}_{t_j}$, which output normalized batch sizes ranged between 0 and 1. The batch size $h_{i,j}$ for each worker i and task j is computed by

$$h_{i,j} = \lceil p\mathbf{y}_{i,t_j} \rceil$$

where $\lceil \rceil$ is the ceiling function. This ensures that the load constraints $\mathcal{C}_1, \mathcal{C}_2$ are satisfied. Each hidden layer consists of 64 units with ReLU activation function [59]. An illustration of the designed DNN is shown in Fig. 3.

To approximate the Q functions, we design a DNN also with four fully connected layers. It takes both state $\mathbf{s}_{t_j}$ and action $\mathbf{a}_{t_j}$ as the input and has a single linear unit in the output layer to generate the Q value. The hidden layers consist of 64 ReLU units.

D. Training DRL Agent

To estimate the parameters in the DNN-based policy and Q functions, we adopt the offline training procedure in [56], which involves the following three stages.

1) Initialization: In the initialization stage (Lines 1-6 in Alg. 1), the weights of the DNNs are randomly initialized. The learning agent then executes the policy constructed using the initial weights for $initial\_episode\_number$ episodes to initialize the replay buffer $\mathcal{D}$. After that, the exploration and parameter update stages described below are performed for $max\_training\_iteration$ iterations, where $max\_training\_iteration$ is a constant large enough for ensuring convergence.

2) Training Data Collection: In the training data collection stage (Lines 8-12 in Alg. 1), the learning agent interacts with the environment to collect the transition data $(\mathbf{s}_{t_j}, \mathbf{a}_{t_j}, r, \mathbf{s}_{t_{j+1}})$ by running the policy constructed with the current weights for $max\_episode\_number$ episodes. In particular, in time step t_j for each task j, given the current state $\mathbf{s}_{t_j}$, the action is generated by applying the policy $\mathbf{a}_{t_j} = \mu(\mathbf{s}_{t_j})$. The next state $\mathbf{s}_{t_{j+1}}$ is obtained by observing the positions $\mathbf{p}_{i,t_{j+1}}$ and velocities $\mathbf{v}_{i,t_{j+1}}$ of all UAVs at time t_{j+1}. The movement of each UAV is described by the mobility model $\mathbf{p}_{i,t_{j+\Delta T}} = f_i(\mathbf{p}_{i,t}, \mathbf{v}_{i,t}, \Delta T)$ with $\mathbf{v}_{i,t_j}$ being the mobility control input. It is noted that our method does not require knowledge of the mobility model and f_i can take any form. With $\mathbf{p}_{i,t_{j+1}}$, the distances $d_{i,t_{j+1}}$ in the next state $\mathbf{s}_{t_{j+1}}$ can then be computed. The rewards r are obtained by applying the reward function described in Sec. V-A4.

The collected transition data is stored in the replay buffer $\mathcal{D}$. After that, a mini-batch $\mathcal{B}$ is randomly sampled from the replay buffer $\mathcal{D}$, which will be used in the next stage to update the parameters of the policy and Q functions.

3) Parameter Update: The parameters θ of the Q function are updated by minimizing the following temporal-difference error:

$$J(\theta) \approx \frac{1}{|\mathcal{B}|} \sum_{(\mathbf{s},\mathbf{a},\mathbf{s}',r)\in\mathcal{B}} \left[L(\mathbf{s}', r) - Q(\mathbf{s}, \mathbf{a}; \theta)\right]^2$$
$$L(\mathbf{s}', r) = r + \gamma Q'(\mathbf{s}', \mu'(\mathbf{s}'; \phi); \theta) \quad (5)$$

The policy parameters ϕ are updated using gradient ascent based on the policy gradient theorem, with the gradient given as follows [57]:

$$\nabla_\phi J(\phi) \approx \frac{1}{|\mathcal{B}|} \sum_{(\mathbf{s},\mathbf{a},\mathbf{s}',\mathbf{r})\in\mathcal{B}} \nabla_\phi \mu(\mathbf{s}; \phi) \nabla_a Q(\mathbf{s}, \mathbf{a}; \theta) \quad (6)$$

To update the parameters, ϕ' and θ', in the target policy and Q functions, Polyak averaging given below is applied:

$$\phi' \leftarrow \tau\phi' + (1-\tau)\phi$$
$$\theta' \leftarrow \tau\theta' + (1-\tau)\theta \quad (7)$$

where $\tau \in (0, 1)$ is a hyperparameter. The complete training procedure is summarized in Alg. 1 and illustrated in Fig. 4.

Algorithm 1: DRL Training Procedure

```
   // Initialization
1  Initialize $\phi, \theta, \theta', \phi', \mathcal{D}$
2  for $k = 1 : initial\_episode\_number$ do
3      for $j = 1 : K$ do
4          Select $\mathbf{a}_{t_j} = \mu(\mathbf{s}_{t_j}; \phi)$.
5          Execute action $\mathbf{a}_{t_j}$ and receive new state $\mathbf{s}_{t_{j+1}}$
             and reward $r$.
6          Store $(\mathbf{s}_{t_j}, \mathbf{a}_{t_j}, r, \mathbf{s}_{t_{j+1}})$ in replay buffer $\mathcal{D}$.
7  for $iteration = 1 : max\_training\_iteration$ do
       // Training Data Collection
8      for $k = 1 : max\_episode\_number$ do
9          for $j = 1 : K$ do
10             Select $\mathbf{a}_{t_j} = \mu(\mathbf{s}_{t_j}; \phi)$.
11             Execute action $\mathbf{a}_{t_j}$ and receive new state
                 $\mathbf{s}_{t_{j+1}}$ and reward $r$.
12             Store $(\mathbf{s}_{t_j}, \mathbf{a}_{t_j}, r, \mathbf{s}_{t_{j+1}})$ in replay buffer $\mathcal{D}$.
13     Sample a random mini-batch $\mathcal{B}$.
       // Parameter Update
14     Update $\theta$ by minimizing the temporal-difference
         error in (5).
15     Update $\phi$ using gradient ascent, with the gradient
         provided in (6).
16     Update $\theta$ and $\phi$ using (7).
```

E. Convergence and Complexity Analysis

Our DRL-based algorithm follows the standard DDPG training procedure, which is not theoretically guaranteed to converge in its general form [60]. In this study, we evaluate the convergence of the proposed algorithm empirically through simulation studies in Section VII.

The time complexity of the proposed DRL-based algorithm is dominated by the training of actor and critic neural networks [61], [62], which is captured by $O(\sum_{j=1}^{J-1} \chi_j \chi_{j+1} + \sum_{z=1}^{Z-1} \hat{\chi}_z \hat{\chi}_{z+1})$. Here J and Z are the number of layers in the actor and critic neural networks, respectively. χ_j, and $\hat{\chi}_z$ represent the number of units in j-th and z-th layer, respectively.

The space complexity of our algorithm is determined by the amount of memory required to store the actor and critic neural networks as well as the replay buffer [61], which is captured by $O(\sum_{j=1}^{J-1} \chi_j \chi_{j+1} + \sum_{z=1}^{Z-1} \hat{\chi}_z \hat{\chi}_{z+1} + |\mathcal{D}|)$.

In our design, $J = 4, Z = 4$, $\chi_1 = \dim(\mathbf{s}), \chi_2 = \chi_3 = 64, \chi_4 = \dim(\mathbf{a}), \hat{\chi}_1 = \dim(\mathbf{s}) + \dim(\mathbf{a}), \hat{\chi}_2 = \hat{\chi}_3 = 64, \hat{\chi}_4 = 1$, and $|\mathcal{D}| = 10^5$, where $\dim(\cdot)$ finds the dimension of a vector.

VI. DRL-based Solution to $\mathcal{P}_2$

In this section, we solve problem $\mathcal{P}_2$ in (3) by extending the DRL method described in the previous section.

A. RL based Formulation for $\mathcal{P}_2$

Similarly, we first model problem $\mathcal{P}_2$ as a MDP characterized by the following quintuple $(\mathbf{s}_t, \mathbf{a}_t, \zeta, r, \mu)$.

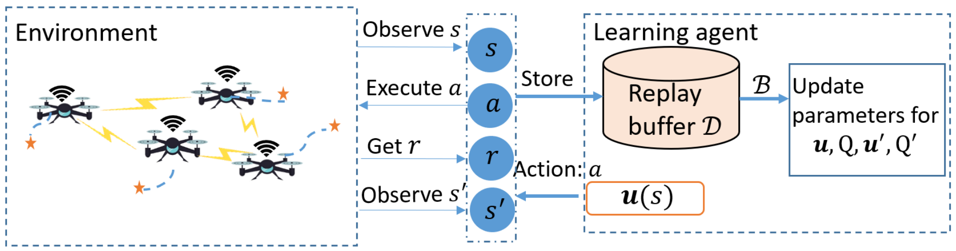


Fig. 4: The training process of the DRL-based method for NAC with uncontrollable UAVs.

1) State $\mathbf{s}_t$*:* As UAVs' velocities are control variables in $\mathcal{P}_2$, we define the state at time t to only include distances $d_{i,t}$ between each worker node i and the master node. The state $\mathbf{s}_t$ is then represented by

$$\mathbf{s}_t = [d_{1,t}, d_{2,t}, \ldots, d_{N,t}]^\top \in \mathcal{S}$$

where $\mathcal{S}$ is the state space.

2) Action $\mathbf{a}_t$*:* In this setting, in addition to the batch size $h_{i,j}$ for each worker node i and task j, the action $\mathbf{a}_{t_j}$ taken at time t_j (start time of task j) also includes the velocities of each node $\mathbf{v}_{i,j}$, i.e.,

$$\mathbf{a}_{t_j} = [h_{1,j}, \ldots, h_{N,j}, \mathbf{v}_{1,j}, \ldots, \mathbf{v}_{N+1,j}]^\top \in \mathcal{A}$$

3) Transition function ζ*:* The explicit form of the transition function ζ is still unknown in $\mathcal{P}_2$. Different from $\mathcal{P}_1$, in order to obtain the next state $\mathbf{s}_{t_{j+1}}$, we should let the NAC system execute task j and UAVs move at the same time based on the action $\mathbf{a}_{t_j}$. $\mathbf{s}_{t_{j+1}}$ can then be obtained by observing the positions of the UAVs at time $t_{j+1} = t_j + T_j$.

4) Reward function r*:* $\mathcal{P}_2$ aims to minimize the weighted sum of the total task completion time $\sum_{j=1}^{K} T_j$ and the total flight time $\sum_{i=1}^{N+1} T_i^{flight}$. To achieve this goal, given current state $\mathbf{s}_{t_j}$ and action $\mathbf{a}_{t_j}$, we define the reward function as follows

$$r(\mathbf{s}_{t_j}, \mathbf{a}_{t_j}) = -\omega_1 T_j - \sum_{i=1}^{N+1} ||\mathbf{p}_{i,t_j} - \mathbf{g}_i||$$

where the first term penalizes long task completion time, and the second term drives each UAV to move closer to its target location and hence arrive there sooner. Moreover, as UAVs should keep a safe distance between each other (constraint $\mathcal{C}_8$ in $\mathcal{P}_2$), we add a third term into the reward function to achieve collision avoidance. The revised reward function is then given by

$$r(\mathbf{s}_{t_j}, \mathbf{a}_{t_j}) = -\omega_1 T_j - \sum_{i=1}^{N+1} ||\mathbf{p}_{i,t_j} - \mathbf{g}_i|| - \omega_c \mathbb{1}_{|\mathbf{p}_{i,t} - \mathbf{p}_{k,t}| \le \epsilon, \forall i,k \in [N+1], i \ne k} \quad (8)$$

where $\omega_c > 0$ is the weight.

5) Policy function μ*:* Similar as $\mathcal{P}_1$, a deterministic policy function $\mu : \mathcal{S} \to \mathcal{A}$ is considered and used to generate the action at the start time t_j of each task $j \in [K]$.

With the above MDP setting, we can then follow the similar procedure described in Sec. V-A to formulate the RL problem.

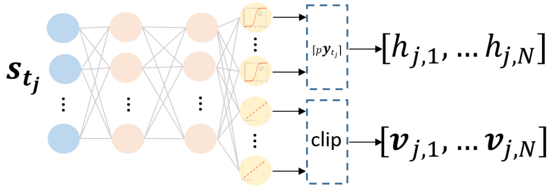


Fig. 5: DNN representation of policy function μ for joint computation load and UAV mobility optimization.

B. Solution to $\mathcal{P}_2$

The derived RL problem can be solved by using the deterministic policy gradient method described in Sec. V-B. For function approximation, we adopt the same DNN structure to approximate the Q functions. To approximate the policy functions, we design a DNN shown in Fig. 5. It differs from the one shown in Fig. 3 in that the output layer contains additional $2N + 2$ linear units for generating UAV mobility control signals, i.e., $\mathbf{v}_{i,j} \in \mathbb{R}^2, \forall i \in [N+1]$. To meet constraint $\mathcal{C}_9$ in $\mathcal{P}_2$, the values generated by these linear units are clipped if falling out of the range $[\mathbf{v}_{min}, \mathbf{v}_{max}]$.

To ensure each UAV will reach its target location (constraint $\mathcal{C}_6$ in $\mathcal{P}_2$), which can happen before or after it completes all computation tasks, we introduce the following mechanism. In case when the UAV has completed all assigned tasks but hasn't reached its target location yet, the UAV switches to another policy that generates mobility control commands only based on its current state. This policy is trained using a similar DRL method with the reward function defined as

$$r(\mathbf{s}_{t_j}, \mathbf{a}_{t_j}) = -\sum_{i=1}^{N+1} ||\mathbf{p}_{i,t_j} - \mathbf{g}_i|| - \omega_c \mathbb{1}_{|\mathbf{p}_{i',t} - \mathbf{p}_{j',t}| \le \epsilon, \forall i',j' \in [N+1], i' \ne j'}.$$

The DNN used for approximating the policy function is similar to the one shown in Fig. 3 but with only $2N + 2$ linear units for generating UAV velocities. Moreover, the UAV changes its velocity after every ΔT. In other cases when the UAV has reached its target location but still has computation tasks remain to complete, the UAV switches to the policy trained using the method presented in Sec. V for generating the task allocation decisions.

VII. Simulation Studies

In this section, we conduct simulation studies to evaluate the performance of the proposed DRL and D-BPCC based methods for NAC under two different formation scenarios. We first design a simulator for the NAC system, and then describe the benchmark schemes used in the comparative studies. The experiment results are presented at the end. All experiments were run on an Alienware Desktop with 32GB memory, 16-cores CPU with 3.6GHz.

A. Simulator Design

To simulate the NAC system, we adopt the following communication, computation and mobility models. It is worth noting that our methods are general and can be applied to NAC systems described by other system models, as no knowledge of the models are required to implement our methods.

1) Communication Model: Suppose all UAVs in the NAC system are equipped with the same directional antennas for long-range and broadband UAV-to-UAV communications, and implement advanced antenna control algorithms to keep the antennas aligned for robust communication [63]. The time to transmit a matrix $\mathbf{X}$ with a rows and b columns between any two UAVs via the UAV-to-UAV link can then be modeled as

$$T^{comm}(\mathbf{X}) = \frac{a \times b \times u}{C}$$

where u (bits) is the average size of the elements in $\mathbf{X}$ and is set as 32 in all experiments. C (bits/sec) is the data rate that can be derived using the Shannon's theory as follows

$$\begin{aligned} C &= W log_2(1 + \frac{S}{N_0}) \\ S &= 10^{\frac{(S_d - 30)}{10}} \\ S_d &= P_t + 20\log_{10}(\lambda) - 20\log_{10}(4\pi) \\ &\quad - 20\log_{10}(d) + G + \kappa \end{aligned}$$

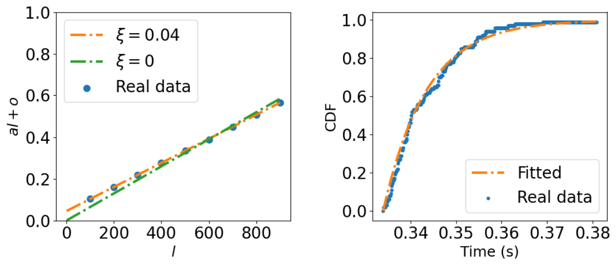
where W (Hz) is the communication bandwidth between two UAVs. N_0 (Watts) is the noise power. S (Watts) is the signal power determined by the transmitting power of the transmitter P_t (dBm), wave length λ (m), sum of the transmitting and receiving gains G (dBi), and distance between the two UAVs d (m). κ denotes the Gaussian noise with zero mean and variance σ [63]. In our simulations, these parameters are configured as $W = 10^4, N_0 = 1.1 \times 10^{-12}, P_t = 27, G = 32, \lambda = 0.12$ and $\sigma = 1$.

2) Computation Model: To simulate the computing power of a UAV, we extend the modeling technique used in many studies [37], [64]. Particularly, we assume the time $T_i^{comp}(\mathbf{A}, \mathbf{x})$ taken by each UAV i to multiply $\mathbf{A} \in \mathbb{R}^{\ell \times m}$ by $\mathbf{x} \in \mathbb{R}^{m \times 1}$ follows a shifted exponential distribution:

$$\mathrm{P}\left[T_i^{comp}(\mathbf{A}, \mathbf{x}) \leq t\right] = 1 - e^{-\frac{\beta_i}{\ell}(t - \alpha_i \ell - \xi_i)} \tag{9}$$

where $t \geq \alpha_i \ell + \xi_i$ specifies the minimum time required to compute the task.

$\beta_i > 0$ and $\alpha_i > 0$ are straggling and shift parameters, respectively, characterizing the computing capability of the UAV. The bias term ξ_i captures the time required for task initialization and function calls. Of note, this term is not included in existing computation models. However, our experiments show that the shifted exponential model with a bias term better captures the characteristics of real computing systems. For illustration purpose, we plot in Fig. 6 the computation model



(a) (b)

Fig. 6: a) Minimum task completion time $\alpha l + \xi$ versus task size l. b) CDF of the task completion time of an Amazon EC2 $t2.xlarge$ instance for computing $\mathbf{A}\mathbf{x}$ with $l = 500$.

constructed for the Amazon EC2 *t2.xlarge* instance by using real experiment data and following the parameter estimation procedure described in [42]. The estimated parameters are $\alpha = 6 \times 10^{-4}, \beta = 1265$ and $\xi = 0.04$.

In the following simulation studies, we let $\xi_i = \xi = 0.04$, $\forall i \in [N]$. The straggling parameter β_i is randomly generated from the range $[100, 500]$ and the shift parameter is set to $\alpha_i = \frac{1}{\beta_i}$ [41].

3) Mobility Model: We assume the UAVs are equipped with an advanced controller robust to wind perturbations and no strong winds are present. The point-mass kinematic model can then be used to simulate the movements of UAVs. In particular, the position of UAV i at time $t + \Delta T$ is computed by the following equation,

$$\mathbf{p}_{i,t+\Delta T} = f(\mathbf{p}_{i,t}, \mathbf{v}_{i,t}, \Delta T) = \mathbf{p}_{i,t} + \mathbf{v}_{i,t}\Delta T$$

In scenarios where the NAC system is formed by uncontrollable UAVs, we let each UAV i change its velocity after each computation task j is completed, with the velocity randomly picked from the range $[-10m/s, 10m/s] \times [-10m/s, 10m/s]$.

With the NAC simulator, we train the proposed DRL methods offline by following the procedure described in Algorithm 1. During the mission, the trained policies generate desired actions online in real time.

B. Benchmarks

We implement the following four representative distributed computing schemes as benchmarks.

1) Uniform Uncoded (UU): In the traditional uncoded distributed computing systems, to perform a matrix-vector multiplication task $\mathbf{A}\mathbf{x}$, the master node decomposes $\mathbf{A} \in \mathbb{R}^{p \times m}$ row-wise into N non-overlapping submatrices $\{\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_N\}$, where $\mathbf{A}_i \in \mathbb{R}^{\ell_i \times m}$, and assigns subtask $\mathbf{A}_i\mathbf{x}$ to work node $i \in [N]$. After receiving results from all worker nodes, the master node can recover $\mathbf{A}\mathbf{x}$ by concatenating the results, i.e., $\mathbf{A}\mathbf{x} = [\mathbf{A}_1\mathbf{x}; \mathbf{A}_2\mathbf{x}; \ldots; \mathbf{A}_N\mathbf{x}]$. To allocate the workload, the UU scheme [37] simply divides the load equally, i.e.,

$$\ell_i = \frac{p}{N}, \forall i \in [N],$$

disregarding the computing power of the worker nodes.

2) Load-Balanced Uncoded (LBU): This scheme [41] divides the computation load according to the computing power of the worker nodes. In particular, the load assigned to each worker node i is inversely proportional to the expected time for this node to compute an inner product, i.e.,

$$\ell_i \propto \frac{\beta_i}{\alpha_i\beta_i+1}, \forall i \in [N]$$

with $\sum_{i=1}^N \ell_i = p$. Note that this scheme requires the knowledge of the computation model.

3) Heterogeneous Coded Matrix Multiplication (HCMM): This is a state-of-the-art CDC scheme for heterogeneous static computing systems [41]. It first encodes matrix $\mathbf{A}$ into a larger matrix $\hat{\mathbf{A}}$ with more rows, and then follows the same procedure as the uncoded schemes to partition and allocate the computation load. The only difference is that the submatrices of the encoded matrix $\hat{\mathbf{A}}$ are multiplied with the input vector at the worker nodes. When the master node receives sufficient results, i.e., the number of rows of aggregated results is no less than p, it can compute the final value. In this scheme, the load assigned to each worker node i is computed by

$$\ell_i = \frac{p}{\lambda_i \eta},$$

where λ_i is the positive solution to $e^{\beta_i\lambda_i} = e^{\alpha_i\beta_i}(\beta_i\lambda_i+1)$, and $\eta = \sum_{i=1}^N \frac{\beta_i}{1+\beta_i\lambda_i}$. Like LBU, HCMM also requires the knowledge of the computation model.

4) Coded Cooperative Computation Protocol (C3P): C3P [39] is a state-of-the-art CDC scheme for heterogeneous mobile computing systems. In this scheme, the master node packetizes each row of $\mathbf{A}$ and encodes each packet. Given an input vector $\mathbf{x}$, it first broadcasts $\mathbf{x}$ to all worker nodes and then gradually offloads the coded packets to the worker nodes one by one. To optimize the computing performance, the offloading interval is dynamically adjusted based on the worker nodes' response times to previous tasks. This scheme does not require any knowledge of the computation, communication or mobility model, and hence can be directly used to solve problem $\mathcal{P}_1$.

As all benchmarks do not consider mobility control, to solve $\mathcal{P}_2$, we apply the benchmarks for load allocation and the DRL method described in Sec. VI-B for UAV mobility control, which runs independently.

C. Evaluation of Solution to $\mathcal{P}_1$

This section evaluates our solution to $\mathcal{P}_1$ for the scenario where the mobility of UAVs is uncontrollable.

1) Experiment Settings: We consider the following three computation scenarios with varying number of UAVs and task sizes.

- **Scenario 1:** $N = 3$, $p = 5000$.
- **Scenario 2:** $N = 6$, $p = 10000$.
- **Scenario 3:** $N = 12$, $p = 20000$.

In all computation scenarios, the dimension of each input vector is set to $m = 10^5$. Initially, the UAVs are randomly distributed over a 400m $\times$ 400m area. The total number of computation tasks to be computed is $K = 25$ and the travel interval is set to $\Delta T = 10s$. To understand the impact of the bias term ξ in the computation model, we also evaluate the case when $\xi = 0$, in addition to the more realistic case when $\xi = 0.04$. In all experiments, the parameters of the DRL method are configured as $\gamma = 0.95, \tau = 0.01, \omega_1 = 15, \omega_c = 5$, $initial\_episode\_num = 205$, $max\_training\_iteration = 1000$, and $max\_episode\_num = 4$.

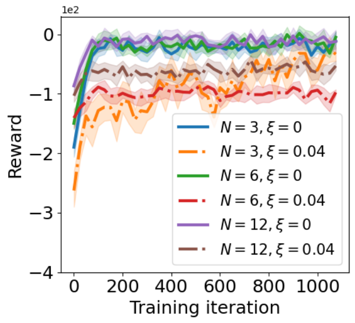


Fig. 7: Training reward of our method for $\mathcal{P}_1$.

2) Training Reward: We first show the learning curves of our method in different computation scenarios with different bias settings. As shown in Fig. 7, our method converges in all scenarios.

3) Comparative Results: The first comparative study evaluates the computation efficiency of different methods. For each computation scenario, we run each method 100 times and record the mean time spent for completing each computation task, referred to as the *average task completion time*. As shown in Fig. 8, our method achieves the highest efficiency in all computation scenarios. Comparing Fig. 8(a) and Fig. 8(b), we can observe that the efficiency of both our method and C3P is significantly impacted by the value of ξ, the overhead induced by task initialization and function calls. However, ξ has a negligible impact on the performance of HCMM, LBU and UU. This is because the worker nodes in our method and C3P process each task batch by batch, where each packet in C3P can be considered as a batch with size $h_{i,j} = 1$, $\forall i \in [N], \forall j \in [K]$. Nevertheless, in HCMM, LBU, and UU, worker nodes process each task as a whole. Hence, when ξ is non-zero, the overhead induced by the many batches in our method and C3P can be significant.

Moreover, we can observe from Fig. 8(a) that although the efficiency of C3P is comparable to our method when the batch overhead is negligible, it is much slower than our method when the batch overhead cannot be ignored (see Fig. 8(b)). This is because the C3P fixes the batch size to 1 and hence does not address the performance-cost trade-offs. Furthermore, as the C3P applies a simple moving average algorithm [39] to approximate the worker nodes' computation times when making the offloading decisions, it achieves a poor performance when the computation times have a large variance, which can happen if UAVs are conducting many other tasks at the same time. To illustrate this, we let the computing parameters μ and α of each worker node change frequently over time, by randomly sampling a new value from the range $[100, 500]$ after each batch is processed. The results are shown in Fig. 9. By comparing Fig. 8(a) and Fig. 9, we can observe that though the performance of both C3P and our method degrades as nodes' computing resources change frequently, our method is much more resilient to such changes. Of interest, UU is not impacted by such changes. This is because UU divides the load equally, disregarding the different computing capabilities of the worker nodes.

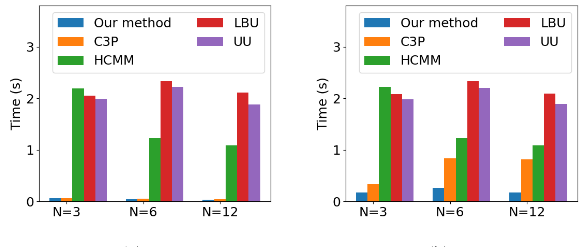


Fig. 8: Average task completion times of different methods in different scenarios when a) $\xi = 0$ and b) $\xi = 0.04$.

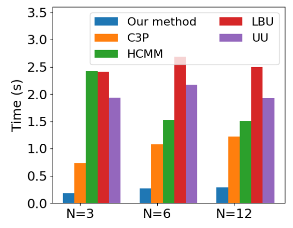


Fig. 9: Average task completion times of different methods in different scenarios when $\xi = 0$ and computation times have a large variance.

The second comparative study evaluates the resilience of different methods to network topology changes caused by high UAV mobility. Particularly, we randomly pick one or two worker nodes and make them move out of the communication range of the master node, which is set to $1500m$. Therefore, results computed by the nodes left cannot be received by the master node.

Fig. 10 shows the simulation results with $\xi = 0.04$. As we can see, our method still achieves the highest efficiency and is the most resilient to topology changes. Of note, there is no data for the LBU and UU schemes, as they require results from all the worker nodes to successfully complete a computation task and hence any node leaving would cause the whole task to fail. To further evaluate the resilience of different methods, we perform a stress test and measure the *success rate* (ratio of successful runs) of each method when the number of nodes left increases. Fig. 11 shows the results for computation Scenario 2 with $N = 6$. The results for the other two scenarios are similar and thus are eliminated to save space. From the figure, we can see that both our method and C3P can complete all computation tasks as long as there is a worker node within the network. The success rate of HCMM decreases quickly when more nodes leave the network, and both UU and LBU fail all tasks when there is one or more nodes left.

D. Evaluation of Solution to $\mathcal{P}_2$

This section evaluates our solution to $\mathcal{P}_2$ for the NAC formation scenario with controllable UAVs.

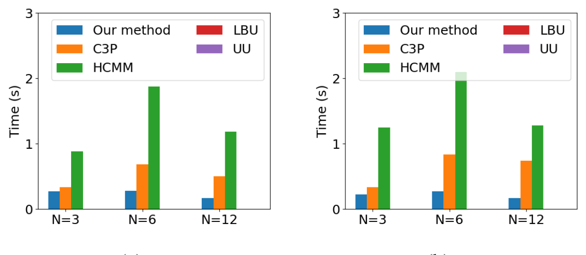


Fig. 10: Average task completion times of different methods in different scenarios when there are a) one and b) two worker nodes leaving the NAC network.

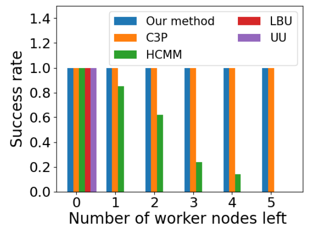


Fig. 11: Success rates of different methods in Scenario 2 ($N = 6$) when an increasing number of worker nodes leave the NAC network.

1) Experiment Settings: We consider the following two computation scenarios.

- **Scenario 1:** $N = 3$, $p = 5000$.
- **Scenario 2:** $N = 6$, $p = 10000$.

The parameters of the reward function in (8) are set to $\omega = 15$ and $\omega_c = 5$. The target locations $\mathbf{g}_i, \forall i \in [N+1]$ are randomly sampled from $[-200m \times 200m] \times [-200m \times 200m]$. The bias term in the computation model is configured as $\xi = 0.04$.

We notice that our DRL method is limited in the scale of the NAC network it can handle, due to the exponentially growing state and action spaces. This is an issue inherent in all RL methods. One potential solution is to use MARL [65], but this requires the change of the computing architecture. We will leave this problem to the future work.

2) Training Reward: Fig. 12 shows the learning curves of our method under different settings. As we can see, the training rewards increase with more training iterations and finally converge.

3) Inference Time: The average inference time of our method measured over 50 runs is about 0.012s, which is small enough for UAVs to promptly react to potential collisions.

4) Comparative Results: Fig. 13(a) shows the total cost J_2 of different methods averaged over 100 experimental runs, where *our method (separate)* refers to the method that optimizes the two objectives of $\mathcal{P}_2$ separately, by using our solution to $\mathcal{P}_1$ for load allocation and the DRL algorithm described in Sec. VI-B for UAV mobility control. As we can see, our method that jointly optimizes the two objectives outperforms all benchmark schemes in achieving the best tradeoff between computation efficiency and travel cost. It is noted that we can tune the weight ω to capture the relative importance of the two objectives depending on the application needs.

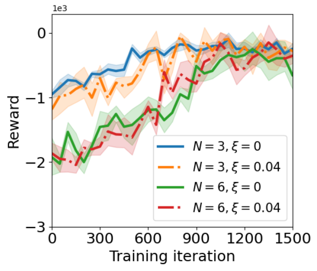


Fig. 12: Training reward of our method for $\mathcal{P}_2$.

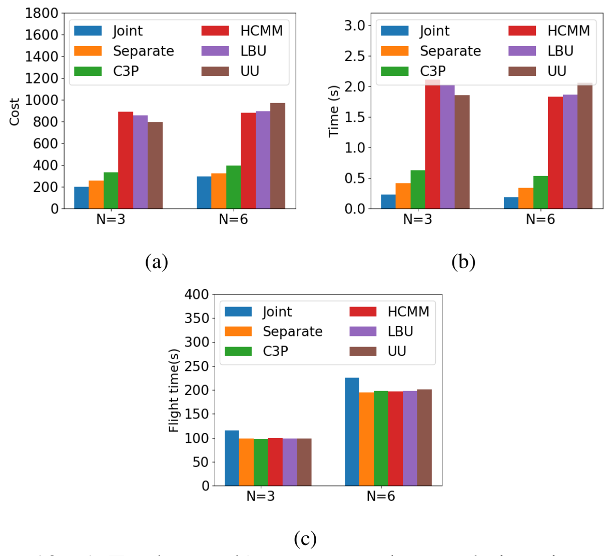


Fig. 13: a) Total cost, b) average task completion times and c) total flight time of different methods in different scenarios.

To better understand the performance of our method, we also plot in Fig. 13(b) and Fig. 13(c) the values of the total task completion time and total flight time, respectively. The results show that our method (joint) completes all computation tasks the most quickly, but consumes the highest UAV flight time. This is expected as all benchmark methods adopt the DRL policy that minimizes the flight time without considering the computing performance. Moreover, the comparison results between our method (joint) and our method (separate) confirm our hypothesis that the mobility of the UAVs can be proactively controlled to facilitate computing.

Fig. 14 plots the sample trajectories of the UAVs in Scenario 1 ($N = 3$) implementing different methods. The initial and target locations of the UAVs are marked using stars and diamonds, respectively. As we can see from Fig. 14(a), our method ensures that all UAVs will reach their target positions. Comparing Fig. 14(a) and Fig. 14(b), it is observed that the trajectories generated by benchmark methods are more straight than that generated by our method. This is because the benchmark methods optimize two objectives separately. Moreover, Fig. 15 illustrates how our method addresses the case when the computation tasks are completed before UAVs arrive at their target locations.

VIII. Conclusion

This paper introduces innovative approaches to enable efficient, robust, and adaptable cooperative airborne computing in the dynamic, heterogeneous, and uncertain airspace. A CDC

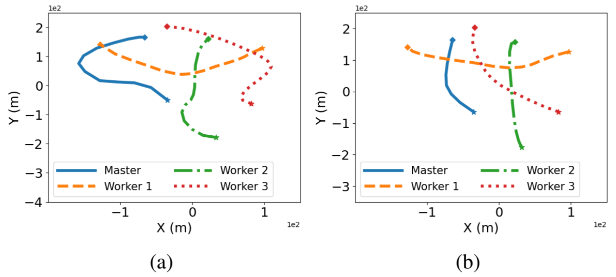


Fig. 14: Sample trajectories of the UAVs in Scenario 1 by using a) our method with joint optimization; and b) benchmark methods.

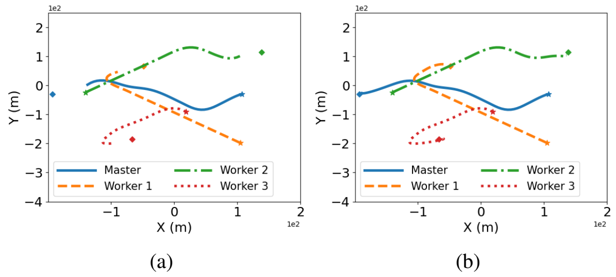


Fig. 15: Sample trajectories of the UAVs in Scenario 1 a) when computation tasks are completed; and b) when the whole mission is completed.

scheme, called D-BPCC, was first introduced that leverages the coding theory and a dynamic batch-processing based procedure to address the uncertainties in the dynamic and heterogeneous NAC system. To optimize system performance, DRL based online decision-making strategies are then designed for two typical NAC formation scenarios, which do not rely on perfect communication, computation or UAV mobility models. Simulation results show that our methods are more resilient to uncertain system disturbances than existing solutions, including the UU, LBU, HCMM, and C3P schemes, and are adaptive to network topology and resource changes. Moreover, the effectiveness of our method in solving scenarios where NAC is formed by controllable UAVs demonstrates the benefits of UAV mobility control to robust computing. In the future, we will investigate MARL to address the scalability issue encountered by our DRL methods when the number of UAVs is large. We will also take energy consumption into the consideration.

Acknowledgement

We would like to thank the National Science Foundation (NSF) under Grants CI-1953048/1730675/1730570/1730325, CAREER-2048266, and CAREER-1714519 for the support of this work.

References

[1] V. Hassija, V. Chamola, A. Agrawal, A. Goyal, N. C. Luong, D. Niyato, F. R. Yu, and M. Guizani, "Fast, reliable, and secure drone communication: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, 2021.

[2] H. Wang, H. Zhao, J. Zhang, D. Ma, J. Li, and J. Wei, "Survey on unmanned aerial vehicle networks: A cyber physical system perspective," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 1027–1070, 2019.

[3] B. Alzahrani, O. S. Oubbati, A. Barnawi, M. Atiquzzaman, and D. Alghazzawi, "Uav assistance paradigm: State-of-the-art in applications and challenges," *Journal of Network and Computer Applications*, vol. 166, p. 102706, 2020.

[4] W. Chen, B. Liu, H. Huang, S. Guo, and Z. Zheng, "When uav swarm meets edge-cloud computing: The qos perspective," *IEEE Network*, vol. 33, no. 2, pp. 36–43, 2019.

[5] K. Lu, J. Xie, Y. Wan, and S. Fu, "Toward uav-based airborne computing," *IEEE Wireless Communications*, vol. 26, no. 6, pp. 172–179, 2019.

[6] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2017.

[7] Q. Hu, Y. Cai, G. Yu, Z. Qin, M. Zhao, and G. Y. Li, "Joint offloading and trajectory design for uav-enabled mobile edge computing systems," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1879–1892, 2018.

[8] F. Zhou, Y. Wu, R. Q. Hu, and Y. Qian, "Computation rate maximization in uav-enabled wireless-powered mobile-edge computing systems," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 9, pp. 1927–1941, 2018.

[9] L. Yang, H. Yao, J. Wang, C. Jiang, A. Benslimane, and Y. Liu, "Multi-uav-enabled load-balance mobile-edge computing for iot networks," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 6898–6908, 2020.

[10] S. Jeong, O. Simeone, and J. Kang, "Mobile edge computing via a uav-mounted cloudlet: Optimization of bit allocation and path planning," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 3, pp. 2049–2063, 2017.

[11] L. Lyu, F. Zeng, Z. Xiao, C. Zhang, H. Jiang, and V. Havyarimana, "Computation bits maximization in uav-enabled mobile edge computing system," *IEEE Internet of Things Journal*, 2021.

[12] 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 Transactions on Vehicular Technology*, vol. 69, no. 3, pp. 3424–3438, 2020.

[13] X. Hu, K.-K. Wong, K. Yang, and Z. Zheng, "Uav-assisted relaying and edge computing: Scheduling and trajectory optimization," *IEEE Transactions on Wireless Communications*, vol. 18, no. 10, pp. 4738–4752, 2019.

[14] J. Xiong, H. Guo, and J. Liu, "Task offloading in uav-aided edge computing: Bit allocation and trajectory optimization," *IEEE Communications Letters*, vol. 23, no. 3, pp. 538–541, 2019.

[15] B. Wang, J. Xie, S. Li, Y. Wan, S. Fu, and K. Lu, "Enabling high-performance onboard computing with virtualization for unmanned aerial systems," in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2018, pp. 202–211.

[16] B. Wang, J. Xie, S. Li, Y. Wan, Y. Gu, S. Fu, and K. Lu, "Computing in the air: An open airborne computing platform," *IET Communications*, vol. 14, no. 15, pp. 2410–2419, 2020.

[17] B. Wang, J. Xie, K. Lu, Y. Wan, and S. Fu, "Coding for heterogeneous uav-based networked airborne computing," in *2019 IEEE Globecom Workshops (GC Wkshps)*. IEEE, 2019, pp. 1–6.

[18] I. Yaqoob, E. Ahmed, A. Gani, S. Mokhtar, M. Imran, and S. Guizani, "Mobile ad hoc cloud: A survey," *Wireless Communications and Mobile Computing*, vol. 16, no. 16, pp. 2572–2589, 2016.

[19] I. Yaqoob, E. Ahmed, A. Gani, S. Mokhtar, and M. Imran, "Heterogeneity-aware task allocation in mobile ad hoc cloud," *IEEE Access*, vol. 5, pp. 1779–1795, 2017.

[20] A. J. Ferrer, J. M. Marquès, and J. Jorba, "Towards the decentralised cloud: Survey on approaches and challenges for mobile, ad hoc, and edge computing," *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, pp. 1–36, 2019.

[21] B. Li, Y. Pei, H. Wu, and B. Shen, "Heuristics to allocate high-performance cloudlets for computation offloading in mobile ad hoc clouds," *The Journal of Supercomputing*, vol. 71, no. 8, pp. 3009–3036, 2015.

[22] B. Zhou, A. V. Dastjerdi, R. N. Calheiros, S. N. Srirama, and R. Buyya, "mcloud: A context-aware offloading framework for heterogeneous mobile cloud," *IEEE Transactions on Services Computing*, vol. 10, no. 5, pp. 797–810, 2015.

[23] N. Fernando, S. W. Loke, and W. Rahayu, "Dynamic mobile cloud computing: Ad hoc and opportunistic job sharing," in *2011 Fourth IEEE International Conference on Utility and Cloud Computing*. IEEE, 2011, pp. 281–286.

[24] L. Liu, C. Chen, Q. Pei, S. Maharjan, and Y. Zhang, "Vehicular edge computing and networking: A survey," *Mobile Networks and Applications*, pp. 1–24, 2020.

[25] W. Zhang, L. Li, N. Zhang, T. Han, and S. Wang, "Air-ground integrated mobile edge networks: A survey," *IEEE Access*, vol. 8, pp. 125 998–126 018, 2020.

[26] Y. Wang, Z.-Y. Ru, K. Wang, and P.-Q. Huang, "Joint deployment and task scheduling optimization for large-scale mobile users in multi-uav-enabled mobile edge computing," *IEEE transactions on cybernetics*, vol. 50, no. 9, pp. 3984–3997, 2019.

[27] Z. Yang, C. Pan, K. Wang, and M. Shikh-Bahaei, "Energy efficient resource allocation in uav-enabled mobile edge computing networks," *IEEE Transactions on Wireless Communications*, vol. 18, no. 9, pp. 4576–4589, 2019.

[28] Y. Luo, W. Ding, and B. Zhang, "Optimization of task scheduling and dynamic service strategy for multi-uav-enabled mobile edge computing system," *IEEE Transactions on Cognitive Communications and Networking*, 2021.

[29] A. Asheralieva and D. Niyato, "Hierarchical game-theoretic and reinforcement learning framework for computational offloading in uav-enabled mobile edge computing networks with multiple service providers," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8753–8769, 2019.

[30] Q. Liu, L. Shi, L. Sun, J. Li, M. Ding, and F. Shu, "Path planning for uav-mounted mobile edge computing with deep reinforcement learning," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 5, pp. 5723–5728, 2020.

[31] L. Wang, K. Wang, C. Pan, W. Xu, N. Aslam, and L. Hanzo, "Multi-agent deep reinforcement learning-based trajectory planning for multi-uav assisted mobile edge computing," *IEEE Transactions on Cognitive Communications and Networking*, vol. 7, no. 1, pp. 73–84, 2020.

[32] X. Tao and W. Song, "Task allocation for mobile crowdsensing with deep reinforcement learning," in *2020 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2020, pp. 1–7.

[33] X. Xiong, K. Zheng, L. Lei, and L. Hou, "Resource allocation based on deep reinforcement learning in iot edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 6, pp. 1133–1146, 2020.

[34] H. Lu, C. Gu, F. Luo, W. Ding, and X. Liu, "Optimization of lightweight task offloading strategy for mobile edge computing based on deep reinforcement learning," *Future Generation Computer Systems*, vol. 102, pp. 847–861, 2020.

[35] O. S. Oubbati, M. Atiquzzaman, A. Baz, H. Alhakami, and J. Ben-Othman, "Dispatch of uavs for urban vehicular networks: A deep reinforcement learning approach," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 12, pp. 13 174–13 189, 2021.

[36] O. S. Oubbati, A. Lakas, and M. Guizani, "Multi-agent deep reinforcement learning for wireless-powered uav networks," *IEEE Internet of Things Journal*, 2022.

[37] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," in *Proc. of IEEE ISIT 2016*. IEEE, aug 2016, pp. 1143–1147.

[38] ——, "Speeding Up Distributed Machine Learning Using Codes," *IEEE Transactions on Information Theory*, vol. 64, no. 3, pp. 1514–1529, mar 2018.

[39] Y. Keshtkarjahromi, Y. Xing, and H. Seferoglu, "Dynamic heterogeneity-aware coded cooperative computation at the edge," in *Proc. of ICNP 2018*. IEEE, sep 2018, pp. 23–33.

[40] ——, "Adaptive and heterogeneity-aware coded cooperative computation at the edge," *IEEE Transactions on Mobile Computing*, 2021.

[41] A. Reisizadeh, S. Prakash, R. Pedarsani, and A. S. Avestimehr, "Coded computation over heterogeneous clusters," *IEEE Transactions on Information Theory*, vol. 65, no. 7, pp. 4227–4242, 2019.

[42] B. Wang, J. Xie, K. Lu, Y. Wan, and S. Fu, "On batch-processing based coded computing for heterogeneous distributed computing systems," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 3, pp. 2438–2454, 2021.

[43] Y. Yang, P. Grover, and S. Kar, "Coded distributed computing for inverse problems," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 709–719.

[44] S. Dutta, V. Cadambe, and P. Grover, "Coded convolution for parallel and distributed computing within a deadline," in *2017 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2017, pp. 2403–2407.

[45] S. Dutta, Z. Bai, H. Jeong, T. M. Low, and P. Grover, "A unified coded deep neural network training strategy based on generalized polydot codes," in *2018 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2018, pp. 1585–1589.

[46] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "Coded mapreduce," in *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 2015, pp. 964–971.

[47] B. Wang, J. Xie, and N. Atanasov, "Coding for distributed multi-agent reinforcement learning," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 10 625–10 631.

[48] K. T. Kim, C. Joe-Wong, and M. Chiang, "Coded edge computing," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 237–246.

[49] A. Frigård, S. Kumar, E. Rosnes *et al.*, "Rateless codes for low-latency distributed inference in mobile edge computing," *arXiv preprint arXiv:2108.07675*, 2021.

[50] A. Asheralieva and D. Niyato, "Fast and secure computational offloading with lagrange coded mobile edge computing," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 5, pp. 4924–4942, 2021.

[51] P. Brusilovski, A. Kobsa, and W. Nejdl, *The adaptive web: methods and strategies of web personalization*. Springer Science & Business Media, 2007, vol. 4321.

[52] Y. Liu, P. Sun, N. Wergeles, and Y. Shang, "A survey and performance evaluation of deep learning methods for small object detection," *Expert Systems with Applications*, vol. 172, p. 114602, 2021.

[53] B. Zhou, J. Xie, and B. Wang, "Dynamic coded convolution with privacy awareness for mobile ad hoc computing," in *2022 IEEE International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2022.

[54] C. Douma, B. Wang, and J. Xie, "Coded distributed path planning for unmanned aerial vehicles," in *AIAA AVIATION 2021 FORUM*, 2021, p. 2378.

[55] Q. Y. Kenny *et al.*, "Indicator function and its application in two-level factorial designs," *The Annals of Statistics*, vol. 31, no. 3, pp. 984–994, 2003.

[56] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[57] R. S. Sutton and A. G. Barto, "An introduction to reinforcement learning, chapter 3," 2018.

[58] A. C. Marreiros, J. Daunizeau, S. J. Kiebel, and K. J. Friston, "Population dynamics: variance and the sigmoid activation function," *Neuroimage*, vol. 42, no. 1, pp. 147–157, 2008.

[59] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

[60] A. Redder, A. Ramaswamy, and H. Karl, "Asymptotic convergence of deep multi-agent actor-critic algorithms," *arXiv preprint arXiv:2201.00570*, 2022.

[61] C. Qiu, Y. Hu, Y. Chen, and B. Zeng, "Deep deterministic policy gradient (ddpg)-based energy harvesting wireless communications," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8577–8588, 2019.

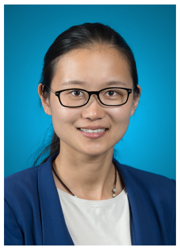
[62] Y. Al-Eryani, M. Akrout, and E. Hossain, "Multiple access in cell-free networks: Outage performance, dynamic clustering, and deep reinforcement learning-based design," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 4, pp. 1028–1042, 2020.

[63] M. Liu, Y. Wan, S. Li, F. L. Lewis, and S. Fu, "Learning and uncertainty-exploited directional antenna control for robust long-distance and broad-band aerial communication," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 1, pp. 593–606, 2019.

[64] A. Reisizadeh, S. Prakash, R. Pedarsani, and S. Avestimehr, "Coded computation over heterogeneous clusters," in *Proc. of IEEE ISIT 2017*, 2017.

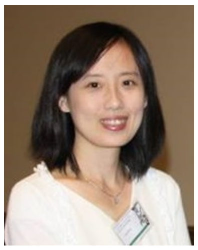
[65] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Advances in neural information processing systems*, 2017, pp. 6379–6390.

Baoqian Wang (S'20) received his B.S. degree from Yangtze University, Wuhan China, in 2017, and M.S. degree in Computer Science from Texas A&M University-Corpus Christi. He is currently a Ph.D. candidate in the joint doctoral program of University of California, San Diego and San Diego State University. His research interests include distributed computing, reinforcement learning and robotics.

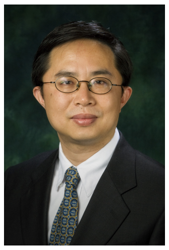


Junfei Xie (S'13-M'16-SM'21) is currently an associate professor in the Department of Electrical and Computer Engineering at San Diego State University. She received the B.S. degree in Electrical Engineering from University of Electronic Science and Technology of China (UESTC), Chengdu, China, in 2012. She received the M.S. degree in Electrical Engineering in 2013 and the Ph.D. degree in Computer Science and Engineering from University of North Texas, Denton, TX, in 2016. From 2016 to 2019, she was an Assistant Professor in the Department of Computing Sciences at Texas A&M University-Corpus Christi. She is the recipient of the NSF CAREER Award. Her current research interests include large-scale dynamic system design and control, airborne networks, airborne computing, and air traffic flow management, etc.

Kejie Lu (S'01-M'04-SM'07) received the BSc and MSc degrees from Beijing University of Posts and Telecommunications, Beijing, China, in 1994 and 1997, respectively, and the PhD degree in electrical engineering from the University of Texas at Dallas, Richardson, Texas, in 2003. In July 2005, he joined the University of Puerto Rico at Mayagüez, Mayagüez, Puerto Rico, where he is currently a professor with the Department of Computer Science and Engineering. His research interests include computer and communication networks, cyber-physical system, and network-based computing.



Yan Wan (S'08-M'09-SM'17) Dr. Yan Wan is currently a Distinguished University Professor in the Electrical Engineering Department at the University of Texas at Arlington. She received her Ph.D. degree in Electrical Engineering from Washington State University in 2009 and then did postdoctoral training at the University of California, Santa Barbara. Her research interests lie in the modeling, evaluation, and control of large-scale dynamical networks, cyber-physical system, stochastic networks, decentralized control, learning control, networking, uncertainty analysis, algebraic graph theory, and their applications to unmanned aerial vehicle (UAV) networking, UAV traffic management, epidemic spread, complex information networks, and air traffic management. Her research has led to over 190 publications and successful technology transfer outcomes.



Shengli Fu (S'01-M'04-SM'07) Shengli Fu received his B.S. and M.S. degrees in telecommunication engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 1994 and 1997, respectively, the M.S. degree in computer engineering from the Wright State University, Dayton, OH, in 2002, and the Ph.D. degree in electrical engineering from the University of Delaware, Newark, DE, in 2005. He is currently a professor and the Chair of the Department of Electrical Engineering, University of North Texas, Denton, TX. His research interests include coding and information theory, wireless communications and sensor networks, and aerial networks.