Energy-Efficient 3D Vehicular Crowdsourcing For Disaster Response by Distributed Deep Reinforcement Learning

Hao Wang
Beijing Inst. of Tech.
Beijing, China
1120171192@bit.edu.cn

Chi Harold Liu
Beijing Inst. of Tech.
Beijing, China
chiliu@bit.edu.cn

Zipeng Dai
Beijing Inst. of Tech.
Beijing, China
3120190984@bit.edu.cn

Jian Tang
Midea Group
Beijing, China
tangjian22@midea.com

Guoren Wang
Beijing Inst. of Tech.
Beijing, China
wanggr@bit.edu.cn

ABSTRACT

Fast and efficient access to environmental and life data is key to the successful disaster response. Vehicular crowdsourcing (VC) by a group of unmanned vehicles (UVs) like drones and unmanned ground vehicles to collect these data from Point-of-Interests (PoIs) e.g., possible survivor spots and fire site, provides an efficient way to assist disaster rescue. In this paper, we explicitly consider to navigate a group of UVs in a 3-dimensional (3D) disaster workzone to maximize the amount of collected data, geographical fairness, energy efficiency, while minimizing data dropout due to limited transmission rate. We propose DRL-DisasterVC(3D), a distributed deep reinforcement learning framework, with a repetitive experience replay (RER) to improve learning efficiency, and a clipped target network to increase learning stability. We also use a 3D convolutional neural network (3D CNN) with multi-head-relational attention (MHRA) for spatial modeling, and add auxiliary pixel control (PC) for spatial exploration. We designed a novel disaster response simulator, called "DisasterSim", and conduct extensive experiments to show that DRL-DisasterVC(3D) outperforms all five baselines in terms of energy efficiency when varying the numbers of UVs, PoIs and SNR threshold.

CCS CONCEPTS

• **Computing methodologies → Distributed artificial intelligence**.

KEYWORDS

Disaster response; vehicular crowdsourcing; energy-efficiency; distributed deep reinforcement learning

ACM Reference Format:
Hao Wang, Chi Harold Liu, Zipeng Dai, Jian Tang, and Guoren Wang. 2021. Energy-Efficient 3D Vehicular Crowdsourcing For Disaster Response by Distributed Deep Reinforcement Learning. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3447548.3467070

KDD '21, August 14–18, 2021, Virtual Event, Singapore

ACM ISBN 978-1-4503-8332-5/21/08…$15.00
https://doi.org/10.1145/3447548.3467070

1 INTRODUCTION

Devastating disasters (e.g., earthquake, hurricane, and explosions) may cause a large amount of missing and casualties in a short period of time. In such urgent situations, rescue teams have to take immediate actions to search as many survivals as possible within the golden 72-hour. To accelerate the disaster response process, an adequate amount of up-to-date data of the entire workzone is highly important.

Spatial crowdsourcing (SC) [9, 10] is a promising technique that has aroused a series of recent industrial successes in data-driven scenarios, including spatiotemporal data collection (e.g., Geo-Wiki and Waze) and urban chauffeuring services (e.g., Uber and DiDi). A typical SC system consists of tasks, workers and the platform. Tasks with spatiotemporal constraints (e.g., deadlines) are submitted to the platform, which performs task assignment to suitable workers. To complete a task, workers (usually with limited energy supply) physically move to the assigned position and submit their collected spatiotemporal data to the platform. The idea of SC is quite promising in assisting quick disaster response by collecting sufficient environmental and survivor-related life data; however, assigning humans as the data collectors is not possible in extreme conditions like disasters. To this end, we explicitly consider to use a group of unmanned vehicles (UVs, like driverless cars and drones) as workers in disaster response, as a Vehicular crowdsourcing (VC [11]) campaign. For example, up-to-date drones from DJI are equipped with multiple sensors and receivers (including temperature sensors, pressure transducers, surveillance cameras, etc.) They can be quickly deployed over disaster workzone, to provide rapid situational awareness with mapping technology and imagery, help firefighters identify hot spots and assess property damage, search for survivors, etc, by collecting environmental and life data from point-of-interests (PoIs) e.g., a camera shooting at possible survivor spots, a smoke detector at the fire site, etc. UVs' powerful ability and deployment flexibility allow to enter dangerous disaster zones instantaneously and complete the sensing task efficiently

which is impossible for humans. Nevertheless, dynamic UV trajectory planning in these extreme conditions is of sparse research exposure and thus is the focus of this paper.

In this paper, we explicitly consider the problem of routing multiple UVs for disaster response. However, key challenges are: (a) UVs may not fully explore the entire disaster zone as a large 3-dimensional (3D) space, which results in the failure of collecting enough data or crashing into various obstacles (e.g., ruins or collapse after the earthquakes); (b) UVs should learn to cooperate with each other, like work division due to limited time and energy supply; (c) PoIs are located at different altitudes so that UVs should carefully plan their trajectories to visit unevenly distributed PoIs in a sequel. Recent state-of-the-art approaches on asynchronous deep reinforcement learning (DRL) like IMPALA [2] paved the way for deriving a good policy efficiently in large exploration space. Our contribution is three-fold:

(1) We propose "DRL-DisasterVC(3D)", a distributed DRL framework for VC tasks in disaster response. It maintains a multi-actor-one-learner architecture to asynchronously collect training experiences for multiple UVs.
(2) We introduce the repetitive experience replay (RER) to improve learning efficiency, and a clipped target network to increase learning stability, respectively. To better extract spatial features from inputs, we use a 3D convolutional neural network (3D CNN) with multi-head-relational attention (MHRA) for spatial modeling, and add auxiliary pixel control (PC) for spatial exploration.
(3) We designed a novel disaster response simulator, called "DisasterSim", to explicitly bridge model training, testing and visualization processes for multi-UV trajectory planning using Unity, Python and Tensorboard together. Based on this, we conduct extensive experiments and results verify the effectiveness of DRL-DisasterVC(3D) when comparing with five baselines.

The rest of the paper is organized as follows. In Section 2, we review the related work. Section 3 presents the system model and problem definition. In Section 4, we design a distributed DRL framework for multi-UV navigation. We show the simulator design in Section 5, followed by the experimental results in Section 6. Finally, we conclude the paper in Section 7.

2 RELATED WORK

2.1 Spatial Crowdsourcing (SC)

SC attracts much attention and has been studied widely both theoretically (e.g., [9, 18]) and for various industrial applications, including web mapping services [21], ride-hailing services [18], online search and recommendation systems [1], etc. Task assignment (TA) is one of the key issues in SC, which aims to allocate spatial tasks to appropriate workers so that the total weighted value (e.g., time cost) is maximized or minimized. TA can be categorized into offline and online scenarios [19]. For the former, the platform is assumed to know all spatiotemporal information of tasks beforehand, and thus a TA problem is formulated as a matching or a route planning problem. For example, Li *et al.* in [9] proposed a 3D stable spatial matching solution. Ni *et al.* in [16] considered the task dependencies and proposed a game-theoretic approach to find an optimal worker-task routing path. In online scenarios, Liu *et al.* in [12] proposed an efficient threshold-based greedy algorithm. Different from all these, in this paper we explicitly consider the disaster response application by navigating a group of UVs to collect data from PoIs in a 3D space.

2.2 DRL

Reinforcement Learning (RL) has been widely adopted to solve sequential decision-making problems by iteratively interacting with a time-slotted environment E. Action a_t is generated by a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$, where the current state s_t transits to s_{t+1} with a reward r_t. The above procedure is usually formulated as a Markov Decision Process (MDP), defined by a tuple $(\mathcal{S}, \mathcal{A}, R, \Omega, \gamma)$. $\mathcal{S}$ and $\mathcal{A}$ represent the state and action space, respectively; $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the expected immediate reward; $\Omega : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \prod(\mathcal{S})$ is the transition probability from the current state s_t and action a_t to the next state s_{t+1}; γ is the discount factor. The emergence of DRL bridges the way between RL and DNNs which successfully makes up the storage of feature representation. A few representative approaches for discrete action space include DQN [15] and Rainbow [5], as well as A3C [14] and DPPO [4] for continuous control. The state-of-the-art approach for asynchronous DRL is IMPALA[2] which is the core of DeepMind AlphaStar [20] that has reached the top league of StarCraft II without any game restrictions. It is a distributed DRL structure, whose learner gathers MDP tuples from each actor instead of gradients. It completely decouples the work of the learner and actors. Then, to reduce the policy lag of asynchronous actors in their own decision-making process, IMPALA extends the target function y_t to the distributed settings, as:

$$y_t = V_\phi(\boldsymbol{s_t}) + \sum_{i=t}^{t+n-1} \gamma^{i-t} \left(\prod_{j=t}^{i-1} c_j \right) \rho_i \left(r_{i+1} + \gamma V_\phi(s_{i+1}) - V_\phi(s_i) \right), \tag{1}$$

where V_ϕ estimates values directly by the current state $\boldsymbol{s_t}$, and c, ρ are truncated important sampling ratios between the actor's current policy and the learner's latest policy. Note that $c_j = \min\left(\overline{c}, \frac{\pi_\theta(a_j|s_j)}{\mu_{\theta'}(a_j|s_j)}\right)$ and $\rho_i = \min\left(\overline{\rho}, \frac{\pi_\theta(a_j|s_j)}{\mu_{\theta'}(a_j|s_j)}\right)$, where $\mu_{\theta'}$ and π_θ are the policy network of actor and learner threads, respectively. As a result, IMPALA increases the speed and decreases the instability of DRL training process, simultaneously. We consider IMPALA as the start point of our design, but it is still quite challenging to directly apply it in disaster response by navigating a group of UVs.

3 PROBLEM FORMULATION

3.1 System Model

Without loss of generality, we consider a VC task for disaster response where a set $\mathcal{U} = \{u|1, 2, ..., U\}$ of UVs (e.g., drones and unmanned ground vehicles for rescue) are used to cooperatively collect data from several PoIs $\mathcal{P} = \{p|1, 2, ..., P\}$, which are known as a prior, in a 3D workzone $\mathcal{L} = \{(x, y, z)|0 < x < L_x, 0 < y < L_y, 0 < z < L_z\}$, where L_x, L_y and L_z denote the maximum length along three dimensions, respectively. A set of obstacles (e.g., collapsed buildings and ruins, etc.) exist that UVs should avoid crashing into. The task duration is fixed, which can be divided into T equal

timeslots of length τ. We assume a timeslot duration τ consists of two parts, UV movement $\tau^u_{t,m}$ and data collection $\tau^u_{t,c}$.

3.1.1 UV Movement. At timeslot t, let (x^u_t, y^u_t, z^u_t) denote the position of UV u, then it moves to a new position at $t+1$ according to angle vector $\boldsymbol{\vartheta}^u_t = \{\vartheta^u_t(pol), \vartheta^u_t(azi)\}$ and moving distance l^u_t, where polar angle $\vartheta^u_t(pol) \in [0, \pi]$, azimuthal angle $\vartheta^u_t(azi) \in [-\pi, \pi]$ and moving distance $l^u_t \in [0, l_{\max}]$, and $l_{\max}$ denotes the maximum distance that a UV can travel in a timeslot. Assume that a UV's movement velocity μ is fixed, then $\tau^u_{t,m} = l^u_t/\mu$.

3.1.2 Data Collection. We assume to use a round robin sensing policy to collect data from $|\overline{\mathcal{P}}^u_t|$ amount of nearest PoIs $\overline{\mathcal{P}}^u_t$, each of which lasts for $\tau^u_{t,c}/|\overline{\mathcal{P}}^u_t| = (\tau - \tau^u_{t,m})/|\overline{\mathcal{P}}^u_t|$. We also assume that PoIs are equipped with multiple antennas operating at orthogonal frequencies, so that data upload transmissions from different PoIs will not interfere with each other (e.g., by OFDMA in 802.11ax with beamforming). Therefore, we only consider the large scale pathloss effect [22] $\omega^{u,p}_t$ between PoI p and UV u as:

$$\omega^{u,p}_t = 20 \log\left(\frac{4\pi f l(u,p)}{c}\right) + \Psi_{NLoS} + \frac{\Psi_{LoS} - \Psi_{NLoS}}{1 + \alpha_1 e^{-\alpha_2 \vartheta(u,p)}}, \quad (2)$$

where f is the channel frequency, c is light velocity, Ψ_{NLoS}, Ψ_{LoS}, α_1, α_2 are constant values which depend on the environment (e.g., rural, urban, dense urban, etc.) $l(u,p)$ and $\vartheta(u,p)$ are the Euclidean distance and elevation angle between PoI p and UV u, respectively. Assuming that the transmission (tx) power of PoIs Ψ_{tx} and average noise power Ψ_n are fixed, the received signal-to-noise-ratio (SNR) denoted by snr can then be calculated by $snr^{u,p}_t = \Psi_{tx} - \omega^{u,p}_t - \Psi_n$. The tx rate from PoI p to a UV u in bps can be represented as: $v^{u,p}_t = W \log(1 + snr^{u,p}_t)$, where W is the bandwidth.

In this case, the amount of successfully uploaded data depends on three factors: data collection time $\tau^u_{t,c}$, tx rate $v^{u,p}_t$, and number of serviced PoIs $|\overline{\mathcal{P}^u_t}|$. If $snr^{u,p}_t < snr_0$ (where snr_0 denotes the threshold), the data cannot be successfully decoded, which results in data dropout $d^{u,p}_{t-}$. Therefore, the received data $d^{u,p}_t$ is:

$$d^{u,p}_t = \begin{cases} 0, & \text{if } snr^{u,p}_t < snr_0 \\ \min\left(d^p_t, v^{u,p}_t \cdot \frac{\tau^u_{t,c}}{|\overline{\mathcal{P}}^u_t|}\right), & \text{otherwise,} \end{cases} \quad (3)$$

where d^p_t denotes the remaining amount of data at PoI p at timeslot t, and $d^{u,p}_{t-} = \min\left(d^p_t, v^{u,p}_t \cdot \frac{\tau^u_{t,c}}{|\overline{\mathcal{P}}^u_t|}\right) - d^{u,p}_t$.

3.1.3 Energy Consumption Model. At the beginning of a task, each UV is fully charged of e_0 energy, and its energy consumption $e^u_t = \beta_1 \mu \tau^u_{t,m} + \beta_2 \tau^u_t$, where the first part is the movement cost, and the second is the data collection cost; β_1, β_2 denote the average energy consumption per meter of traveling, and per second of data collection, respectively.

3.2 Problem Definition

We introduce four metrics to mathematically define our UV navigation problem for disaster response.

Data Collection Ratio ζ: it is defined as the average ratio between collected data from all UVs and initial data reserve of all PoIs when the task is completed, as:

$$\zeta = \frac{\sum_{p=1}^{P} d^p}{\sum_{p=1}^{P} d^p_0}, \quad (4)$$

where $d^p = \sum_{t=1}^{T} \sum_{u=1}^{U} d^{u,p}_t$ is the total amount of collected data from PoI p by all UVs at the end of a task.

Data Dropout Ratio σ: to measure the quality of data collection process due to impact of low SNR, by computing the percentage of data loss, as:

$$\sigma = \left(1 + \frac{\sum_{t=1}^{T} \sum_{u=1}^{U} \sum_{p \in \overline{\mathcal{P}}^u_t} d^{u,p}_t}{\sum_{t=1}^{T} \sum_{u=1}^{U} \sum_{p \in \overline{\mathcal{P}}^u_t} d^{u,p}_{t-}}\right)^{-1}. \quad (5)$$

Geographical Fairness κ: for sake of pursing data diversity and uniformity in the 3D disaster workzone, we define a metric called "geographical fairness κ" by using the Jain's fairness index [8]:

$$\kappa = \frac{\left(\sum_{p=1}^{P} \frac{d^p_0 - d^p_t}{d^p_0}\right)^2}{P \sum_{p=1}^{P} \left(\frac{d^p_0 - d^p_t}{d^p_0}\right)^2}. \quad (6)$$

Energy Efficiency ξ: to achieve the above three goals simultaneously, we define one single metric to combine them in an energy efficient way to measure the overall performance of our task, as:

$$\xi = \frac{\zeta \cdot \sum_{p=1}^{P} d^p_0}{e_T} \cdot (1 - \sigma) \cdot \kappa, \quad (7)$$

where $e_T = \sum_{t=1}^{T} \sum_{u=1}^{U} e^u_t$ denotes the total energy consumption of all UVs. Then, the optimization problem is formulated as:

$$\boldsymbol{P1}: \max_{\{\boldsymbol{\vartheta}^u_t, l^u_t\}} \xi, \quad s.t. \sum_{t=1}^{T} e^u_t \le e_0, \forall u \in \mathcal{U}. \quad (8)$$

Note that $\boldsymbol{P1}$ is challenging to solve due to the following reasons. First, data associated with PoIs are unevenly distributed in the 3D workzone, thus fully exploring the whole space while collecting enough data and avoiding crashing into obstacles is quite difficult to achieve. Second, UVs need to learn to cooperate due to their limited energy supply. Proper cooperation can reduce invalid movement while increasing energy efficiency. Finally, how to balance geographical fairness and energy consumption becomes an issue, since some PoIs are far-off which are hard to visit. Obviously, $\boldsymbol{P1}$ is NP-hard, which is difficult to solve in general but naturally appropriate for heuristic methods by DRL. We model $\boldsymbol{P1}$ as a MDP $\mathcal{M} =< \mathcal{S}, \mathcal{A}, R, \Omega, \gamma >$.

State Space $\mathcal{S}$: let $\mathcal{S} \triangleq \{\boldsymbol{s}_t = (\mathcal{S}_1, \mathcal{S}_2)\}$ denote the state information which includes two channels and each channel is a 3D vector. As shown in Figure 1, we discretize the entire workzone into $50 \times 50 \times 10$ unit-size cubes. $\mathcal{S}_1$ includes the coordinates of UVs and obstacles. In $\mathcal{S}_2$, we place the remaining energy of UV u at its current position (x^u_t, y^u_t, z^u_t) and the current remaining data d^p_t of PoI p at (x^p_t, y^p_t, z^p_t).

Action Space $\mathcal{A}$: consists of two parts $\mathcal{A} \triangleq \{\boldsymbol{a}_t = (\boldsymbol{\vartheta}^u_t, l^u_t)_{u \in U}\}$. Here $\boldsymbol{\vartheta}^u_t$ is the angle vector that shows the vertical and horizontal movement directions; and l^u_t is the traveling distance, which

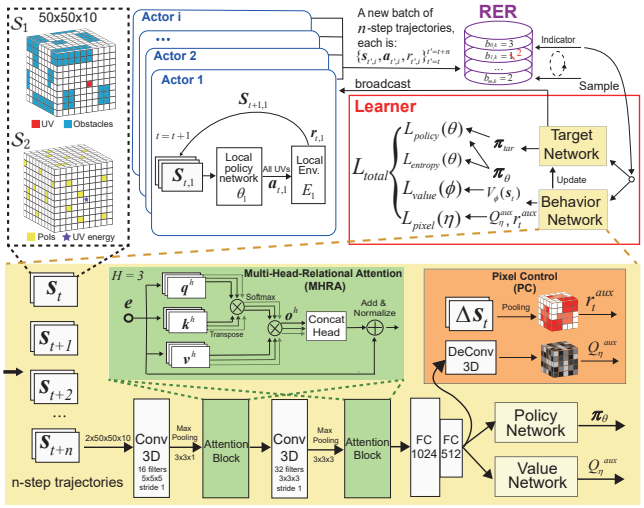


Figure 1: Proposed solution: DRL-DisasterVC(3D).

cannot go over the maximum distance $l_{\max}$. For simplicity, we discretize each action into equal number of divisions in its value range and then we can transform the continuous control problem into a discrete one.

Reward Function: it is defined as:

$$r_t = \left(\frac{1}{U} \sum_{u=1}^{U} \frac{d_t^u}{e_t^u} \cdot \left(1 - \frac{d_{t-}^u}{d_t^u + d_{t-}^u}\right) \right) \cdot \kappa_t - \varrho_t. \tag{9}$$

Here $d_t^u = \sum_{p \in \overline{\mathcal{P}}_t^u} d_t^{u,p}$ and $d_{t-}^u = \sum_{p \in \overline{\mathcal{P}}_t^u} d_{t-}^{u,p}$ are the amount of collected and dropped data by UV u at timeslot t, respectively. κ_t is the time-varying geographical fairness index, which is similar to (6). ϱ_t denotes the penalty applied to UVs, when hitting an obstacle or running out of energy.

4 SOLUTION: DRL-DISASTERVC(3D)

Due to the scale and complexity of solution space in ***P1***, we opt to propose a heuristic method by DRL called "DRL-DisasterVC(3D)". It consists of a distributed, asynchronous DRL framework based on IMPALA (see Section 4.1), and the attentive 3D CNN with auxiliary pixel control for spatial exploration (see Section 4.2).

4.1 Distributed DRL Framework with RER for Multi-UV Planning in Disaster Response

As shown in Figure 1, we adopt the multi-actor-one-learner architecture from IMPALA [2] as the start point of our design, where asynchronous actors work on different GPUs to generate MDP tuples by interacting with their local environment E_i. The learner collects these training data and then updates DNN parameters by gradient descent methods. Although IMPALA can collect millions of MDP tuples, it uses each MDP tuple only once. The quality of learning from sampled experiences determines the training accuracy and speed, which is critical in disaster response. To address this, we introduce the repetitive experience replay (RER) for improving learning efficiency, and clipped target network for increasing learning stability, respectively.

4.1.1 Multi-Agent Policy and RER. As shown in Figure 1, the policy network output is a multi-agent policy $\boldsymbol{\pi}_\theta$. To better utilize previous experiences for multiple UVs in IMPALA, and to lead the policy to reach the global optima, we store each batch of experiences in a RER $\mathcal{B}$. It can store up to b_M batches. Each can be traversed at maximum b_K times, with a visit counter. Specifically, we define a zero-initialized visit counter $b_{m,k}$ for the m-th batch at first. After sampling a batch of experiences from $\mathcal{B}$, the visit counter is updated by $b_{m,k} = b_{m,k} + 1$. Then the indicator scrolls to $(m+1) \mod b_M$ where a new batch is sampled again. If any $b_{m,k}$ reaches b_K, the m-th batch will be discarded and replaced by the latest batch. By reusing experiences, sampling efficiency is directly improved and the multi-agent policy $\boldsymbol{\pi}_\theta$ can be fairer for all UVs.

4.1.2 Clipped Target Network for Distributed Learning Stability. In IMPALA, an actor enforces a different local policy $\boldsymbol{\pi}_{act_i}$ and they are updated asynchronously by the learner's behavior network of policy $\boldsymbol{\pi}_\theta$. Compared to the single-UV case, action space $\mathcal{A}$ in our multi-UV scenario expands explosively in dimensions, which enlarges the difference among $\boldsymbol{\pi}_{act_i}$. In order to stabilize the distributed training process, we explicitly add a target network of policy $\boldsymbol{\pi}_{tar}$, which is synchronized by $\boldsymbol{\pi}_\theta$ periodically. However, in a distributed DRL architecture, this off-policy mechanism may bring deviation and instability to the value estimate function V. Thus, we further limit the policy update speed by using truncated importance sampling by $\min(\frac{\boldsymbol{\pi}_{act_i}}{\boldsymbol{\pi}_{tar}}, \rho) \frac{\boldsymbol{\pi}_\theta}{\boldsymbol{\pi}_{act_i}}$, where ρ is the clipped ratio. The agent learns fast when setting ρ a high value at the cost of training instability. Finally, considering an n-step experience $\{\boldsymbol{s}_{t'}, \boldsymbol{a}_{t'}, r_{t'}\}_{t'=t}^{t'=t+n}$, target function y_t is computed by:

$$y_t = V_\phi(\boldsymbol{s}_t) + \hat{A}(n,t), \tag{10}$$

where V and A are estimated value and n-step advantage function, respectively. Evaluating actions in multiple steps brings high variance and bias to y_t. Thus, different from IMPALA using Vtrace in Eqn. (1), we improve $\hat{A}(n,t)$ by GAE-λ[17] with V-trace [3]:

$$\hat{A}(n,t) = \sum_{i=t}^{t+n-1} (\lambda\gamma)^{i-t} \left(\prod_{j=t}^{i-1} c_j \right) \delta_i V, \tag{11}$$

where $c_i = \min(\overline{c}, \frac{\boldsymbol{\pi}_{tar}}{\boldsymbol{\pi}_{act_i}})$, that reduces the high variance in policy gradients and thus reduces interaction frequency with environment, which is beneficial in our disaster response scenario.

During the training process in the learner, we consider all policy, value and entropy losses. We use $\epsilon - clip$ to ensure the reasonable multi-agent policy gradient steps in the trust region. Therefore, at timeslot t, the policy network parameter θ are updated by:

$$\begin{aligned} L_{policy}(\theta) = \mathbb{E}\Bigg[\min\Bigg(& \min\left(\frac{\boldsymbol{\pi}_{act_i}}{\boldsymbol{\pi}_{tar}}, \rho\right) \frac{\boldsymbol{\pi}_\theta}{\boldsymbol{\pi}_{act_i}} \hat{A}(n,t), \\ & clip\Big(\min\left(\frac{\boldsymbol{\pi}_{act_i}}{\boldsymbol{\pi}_{tar}}, \rho\right) \frac{\boldsymbol{\pi}_\theta}{\boldsymbol{\pi}_{act_i}}, 1-\epsilon, 1+\epsilon \Big) \hat{A}(n,t) \Big) \Bigg], \end{aligned} \tag{12}$$

and the value parameters ϕ of shared critic network are updated using l_2 loss function:

$$L_{value}(\phi) = \mathbb{E}[(y_t - V_\phi(\boldsymbol{s}_t))^2]. \tag{13}$$

Then, to encourage UV explorations, we add an entropy bonus:

$$L_{entropy}(\theta) = -\sum_{\boldsymbol{a}_t} \boldsymbol{\pi}_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t) \log \boldsymbol{\pi}_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t). \quad (14)$$

Note that our multi-agent policy considers all UV actions in L_{policy}, L_{value} and $L_{entropy}$.

4.2 Attentive 3D CNN Convolution with Pixel Control for Spatial Exploration

In order to better extract spatial features from inputs, we use a 3D CNN with multi-head-relational attention (MHRA) for spatial modeling, and add auxiliary pixel control (PC) for spatial exploration.

4.2.1 MHRA for Spatial Modeling. As shown in Figure 1, we used two 3D CNN layers to calculate the spatial feature representations of input state. Since certain relationships do exist between different spatial dimensions (e.g., collaboration among UVs in adjacent areas, spatial order of PoIs in collapsed environment). Better extracting these relationships helps UVs learn more reasonable trajectories, by adding a MHRA module between every two 3D CNN layers. Here are two MHRA modules. Take the first layer as an example, considering that the input vector is the first 3D CNN output denoted by $\boldsymbol{e} = \text{3DConv}(\boldsymbol{s}_t)$, we project state vector $\boldsymbol{e}$ into query $\boldsymbol{q}$, key $\boldsymbol{k}$, and value vector $\boldsymbol{v}$, as: Query : $\boldsymbol{q} = f_q(\boldsymbol{e})$, Key : $\boldsymbol{k} = f_k(\boldsymbol{e})$, Value : $\boldsymbol{v} = f_v(\boldsymbol{e})$, where f_q, f_k, f_v are all $1 \times 1 \times 1$ 3D convolution filters. Each q is compared to all unit-cubes' keys $\boldsymbol{k}$ via a dot-product and normalization operation $softmax\left(\frac{k \cdot q}{\sqrt{g}}\right)$, where g is a scaling factor that equals to the dimensionality of $\boldsymbol{k}$, which indicates the correlative degree of attention between a specific unit-cube and all others. For some unit-cubes as an area, the attention outputs are computed by the weighted mixture of all underlying unit-cubes' value vectors, as:

$$O = softmax\left(\frac{\mathcal{Q}\mathcal{K}^\top}{\sqrt{g}}\right)\mathcal{V}. \quad (15)$$

We use H independent attention heads in parallel. Each has its own set of $1 \times 1 \times 1$ convolution filter to produce different queries, keys and values which indicate the different relational semantics. Then, we concatenate all head outputs $\boldsymbol{o}^h$, pass to a fully connected layer with the same layer size as $\boldsymbol{e}$ to produce an output. To speed up model convergence, we use layer normalization and a shortcut to connect the input and output of attention module. Finally, the outputs of the second attention module in Figure 1 are fed to two fully connected layers, and then diverted to value, policy, PC networks, respectively.

4.2.2 Auxiliary PC for Spatial Exploration. Insufficient spatial exploration would lead to low data collection ratio and geographical fairness in ***P1***. This is because that some PoIs located in corner areas are hard to collect, and obstacle distributions are quite random due to the disaster, thus PoI locations are not uniformly distributed, which is challenging for UVs to visit. Therefore, at the beginning of model training, a good policy may encourage UVs to explore all the PoIs regardless of energy consumption. Calculating the difference between adjacent states will reflect certain degree of spatiotemporal feature from geographical fairness and collected data amount in the 3D workzone. Mining this from input pixel differences potentially allows UVs to plan their trajectories more effectively, and we refer this pixel difference as the "intrinsic reward", and adopt a 3D deconvolutional network as auxiliary value function in an unsupervised way. Then, we integrate an unsupervised auxiliary task called "PC" [7], as shown in Figure 1, which runs as follows. First, we define an intrinsic reward $\boldsymbol{r}_t^{aux}$ by calculating the average absolute difference of adjacent input state unit-cubes in a sequel. Then, we use a 3D deconvolutional network with a dueling structure to compute an additional value function Q_t^{aux} from the outputs of the fully connected network. Note that Q_t^{aux} is a 3D spatial grid of action values, which represents the current estimate of expected changes of adjacent states, after each all UVs take action $\boldsymbol{a}_t$ at timeslot t. Finally, we update the deconvolutional network by optimizing a n-step Q-learning loss:

$$L_{pixel}(\eta) = \mathbb{E}[(y_t^{aux} - Q_t^{aux}(\boldsymbol{s}_t, \boldsymbol{a}_t, \eta))^2], \quad (16)$$

where the target Q-value y_t^{aux} is defined as $y_t^{aux} = \sum_{k=1}^{n} \gamma^k r_{t+k}^{aux} + \gamma^n \max_{a'} Q_t^{aux}(\boldsymbol{s}_{t+n}, \boldsymbol{a}_{t+n}, \eta')$.

Algorithm 1: Actor-i

Input: Local environment E_i and policy network θ_i

1 **while** *leaner updates* **do**
2 B_i = empty set;
3 **while** B_i *is not full* **do**
4 **for** *each n-step trajectory* **do**
5 Store $\{\boldsymbol{s}_{t',i}, \boldsymbol{a}_{t',i}, r_{t',i}\}_{t'=t}^{t'=t+n}$ in B_i;
6 Send B_i to RER $\mathcal{B}$;
7 **if** *broadcasted weights θ' exist* **then**
8 update local policy $\theta \leftarrow \theta'$

4.3 Algorithm Descriptions

Our algorithm consists of two parts: multiple actors and a central learner. Actors run in different processes, interacting with their local environment independently and sending a batch of experiences to RER periodically and asynchronously. The central learner samples a batch of experiences from RER and optimizes the network by gradient descent. Pseudocode is given in Algorithm 1 and 2 .

Actors: Several actors generate experiences for the central learner and work asynchronously with a shared RER $\mathcal{B}(b_K, b_M)$ until the learner finishes updating network. Each actor i maintains a small buffer B_i that stores multiple n-step trajectories collected from local environment E_i. When an actor interacts with E_i for n times, it stores the experience $\{\boldsymbol{s}_{t',i}, \boldsymbol{a}_{t',i}, r_{t',i}\}_{t'=t}^{t'=t+n}$ in the batch and starts the next timeslot (Line 4-5). When B_i is full, the actor sends it to the $\mathcal{B}$ (Line 6), together with their corresponding policies $\{\boldsymbol{\pi}_{t'}\}_{t'=t}^{t'=t+n}$, to compute the importance ratio. If the actor receives the broadcasted hyperparamter θ' from the central learner, the local policy is synchronized by $\theta \leftarrow \theta$' (Line 7-8).

Learner: The central learner maintains a RER $\mathcal{B}(b_K, b_M)$ to gather experiences from multiple actors. At the beginning, it randomly initializes the behavior network and copies parameters to the target network (Line 1-2). For every gradient descent step, the learner samples a batch of experiences B_n from RER at the indicator position, then uses Eqn. (12), (13), (14), (16) to compute $L_{policy}(\theta)$,

Algorithm 2: Learner

Input: policy network θ, value network ϕ, PC network η and their target networks θ', ϕ', η'.

1 Randomly initialize network weights (θ, ϕ, η);
2 Initialize target network $(\theta', \phi', \eta') \leftarrow (\theta, \phi, \eta)$;
3 Initialize RER $\mathcal{B}(b_K, b_M)$;
4 **for** *Episode=1,2,3...* **do**
5 Sample batch B_m traversed $b_{m,k}$ times from $\mathcal{B}$;
6 $b_{m,k} = b_{m,k} + 1$;
7 Use Eqn. (12), (13), (14), (16) to compute $L_{policy}(\theta), L_{value}(\phi), L_{entropy}(\theta), L_{pixel}(\eta)$;
8 Calculate L_{total} by weighted sum of all losses;
9 Optimize surrogate L_{total} by gradient descent methods;
10 **if** $b_{m,k} = b_K$ **then**
11 discard batch B_m from $\mathcal{B}$;
12 **if** $t \mod t_{tar} = 0$ **then**
13 update target network $(\theta, \phi, \eta) \leftarrow (\theta', \phi', \eta')$;
14 **if** $t \mod t_{sync} = 0$ **then**
15 broadcast weights to all actors;

$L_{value}(\phi)$, $L_{entropy}(\theta)$ and $L_{control}(\eta)$, respectively. The total loss L_{total} is calculated by weighted sum of all losses. Next, the learner uses gradient descent methods to optimize surrogate L_{total} by optimizing parameters θ, ϕ, η (Line 7-9). After a gradient descent step, the visit counter of used experiences will set to $b_{m,k} = b_{m,k} + 1$. If a batch of experiences used b_K times, the batch is discarded and replaced by a new batch (Line 10-11). Finally, after t_{tar} and t_{sync} period of time, the learner updates the target network and broadcasts network weights to all actors (Line 12-15).

5 DISASTERSIM: A SIMULATOR FOR VC IN DISASTER RESPONSE

5.1 Simulator Design

We design a novel disaster response simulator, called "DisasterSim", to explicitly bridge model training, testing and visualization processes for multi-UV trajectory planning using Unity 2018.3.14f1, Python 3.7.7 and Tensorboard 2.3.0. As shown in Figure 2, it consists of three layers: data layer, model layer and visualization layer. The data layer consists of (a) scene configuration including scene assets (e.g., building models, serialized by Unity Asset Bundle stored in ".ab" format), system parameters like Ψ_{Los}, Ψ_{NLoS} in Eqn. (2) and task settings like P, U, (b) model data (e.g., DNNs structures, optimizer and network parameters ϕ, θ, η) stored in ".pth" file, and (c) a running log. The model layer is composed of four modules: Scene Generator, Hyperparameters Tuning, Model Training, and Testing. First, the Scene Generator loads the specific scene configurations from the data layer and creates a virtual 3D scene by Unity Physics Engines for model training and testing. Then, intermediate results like value loss in Eqn. (13) and reward in Eqn. (9) are dynamically displayed to users on Tensorboard Dashboard, and model weights are saved periodically as a serialized object to the data layer. We also offer a way to observe the model performance intuitively

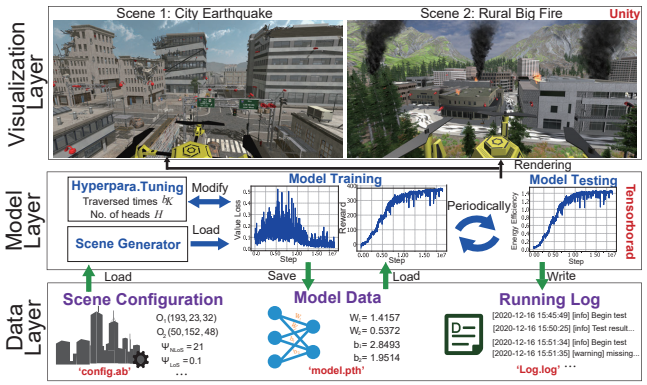


Figure 2: DisasterSim

by rendering UV trajectories on Unity Platform. To find suitable hyperparameters, Hyperparameter Tuning module changes scene configuration and re-run the model training again. All the above operations produce running logs which are saved in the data layer as ".log" file.

5.2 Setting

DisasterSim runs on a Ubuntu 18.04 server with 8 NVIDIA RTX graphic card and Intel(R) Xeon(R) Gold 6238 CPU @2.10GHz. PoIs are deployed in a $1000 \times 1000 \times 200$ meter 3D workzone. Each PoI p is initialized with data amount $d_0^p = 10$Mbit. The initial location of each UV is at $(500, 500, 100)$m with movement speed $\mu = 20$m/s. $T = 400$ and $\tau = 20$ seconds are set in each episode. System parameter for communications are: $\Psi_{LoS} = 0.1$, $\Psi_{NLoS} = 21$, $\alpha_1 = 39.79$, $\alpha_2 = 0.43$. PoI tx power $\Psi_{tx} = 20$dbm and average noise power $\Psi_n = -70$dbm. The channel frequency is 2.4GHz and available bandwidth is $W = 20$MHz. Note that each UV's initial energy reserve is $e_0 = 1500$kJ. Two energy consumption weights are $\beta_1 = 0.075$kJ/m, and $\beta_2 = 0.025$kJ/s. And the number of serviced PoIs for each UV at a timeslot $|\overline{\mathcal{P}}_t^u| = 10$.

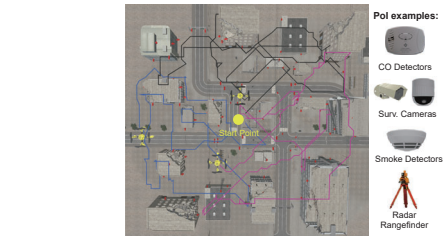
5.3 Visualization

We next visualize moving trajectories of 3 UVs (in particular we use drones) and $P = 175$ in DisasterSim. As shown in Figure 3a, we observe that drones learn to collaborate by roughly dividing the workzone into 3 parts, and move around in its responsible one. This behavior pattern aids to maximize energy efficiency. Also, drones tend to visit PoIs by flying (sometimes circulating) around the exterior of buildings, which helps achieve maximum tx rate (lower pathloss from PoIs to a drone). This is confirmed in Figure 3b as a first-person view, where 3 drones depart from the same start point and then first collect low-altitude data, followed by drone 1 moving up the exterior wall, while drone 2 circulating around a collapsed building to collect its high-altitude data.

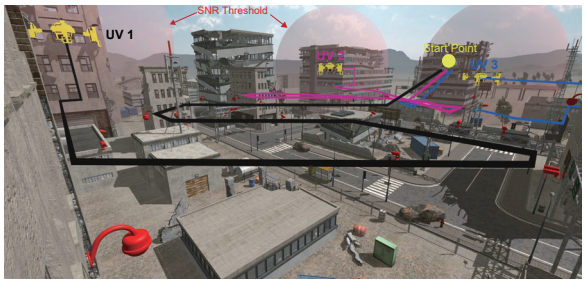
6 EXPERIMENT

6.1 Ablation Study

We perform ablation study by comparing their performance while gradually removing two key components of DRL-DisasterVC(3D), namely: MHRA and PC. As shown in Table 1, we see that: (1) data collection ratio ζ and geographical fairness κ of DRL-DisasterVC(3D)



(a) Trajectory overview



(b) First-person view

Figure 3: Trajectories of 3 drones to collect video and fire/smoke data from 175 PoIs in a city earthquake.

attain 4.5% and 3.9% improvement over the one without PC, which confirms that PC helps to achieve a better spatial exploration; (2) energy efficiency ξ decreases significantly if removing MHRA when compared to the one without PC. This is because that PC sacrifices some degree of ξ to achieve a wider spatial exploration, and MHRA weakens this shortcoming by extracting more spatial relational features; and (3) when removing both PC and MHRA, the complete version is 17.3% better which confirms the benefits of putting MHRA and PC together.

6.2 Hyperparameter Tuning

Suitable hyperparameters in DNNs will significantly improve the overall performance of DRL-DisasterVC(3D). Specially, we consider the number of heads $H = \{1, 2, 4, 8, 16\}$ (in MHRA), and number of traversed times $b_K = \{1, 2, 3, 4\}$ (in RER). For other hyperparameters, we simply used some common settings which are previously used in IMPALA [2], as: learning rate 0.0007, discount factor $\gamma = 0.99$, batch size 150, sequence length 15 and 16 actors. Specially, importance sampling clipped ratio $\rho = 2$, $\lambda = 0.995$, $\epsilon = 0.3$, $B_M = 8$. We consider $U = 2$ UVs, $P = 250$ PoIs, $snr_0 = 15$dbm. During training, network weights are saved periodically. For testing, we run each model for $T = 400$ timeslots and repeat 50 times to take an average. We can make the following observations from Table 2:

(1) With the increase of traversed times b_K for each batch of experiences, energy efficiency ξ is also increased. For example, with 4-head attention module, ξ increases from 1.309 to 1.437. This is because that RER can improve sample efficiency and learning quality from limited experiences. When b_K is too big, ξ starts to drop, since overfitting issue may happen with the shortage of experience diversity which eventually leads to worse performance.

(2) When no. of attention heads H increases, energy efficiency ξ also has an obvious improvement. For example, with $b_K = 3$, ξ increases from 1.238 to 1.437. MHRA aids to extract multi-level relational representations in different semantics, which helps our model to make better decisions. But too many heads leads to a poor performance, due to the difficulties of model convergence incurred by significantly increased number of network parameters.

Therefore, we find that $H = 4$ heads with $b_K = 3$ traversed times give the best performance in terms of energy efficiency that will be used for the rest of performance comparisons.

Table 1: Ablation study

	ζ	σ	κ	ξ
DRL-DisasterVC(3D)	**0.921**	**0.108**	**0.945**	**1.440**
- w/o PC	0.876	0.114	0.906	1.355
- w/o MHRA	0.898	0.119	0.919	1.304
- w/o PC, MHRA	0.842	0.133	0.867	1.227

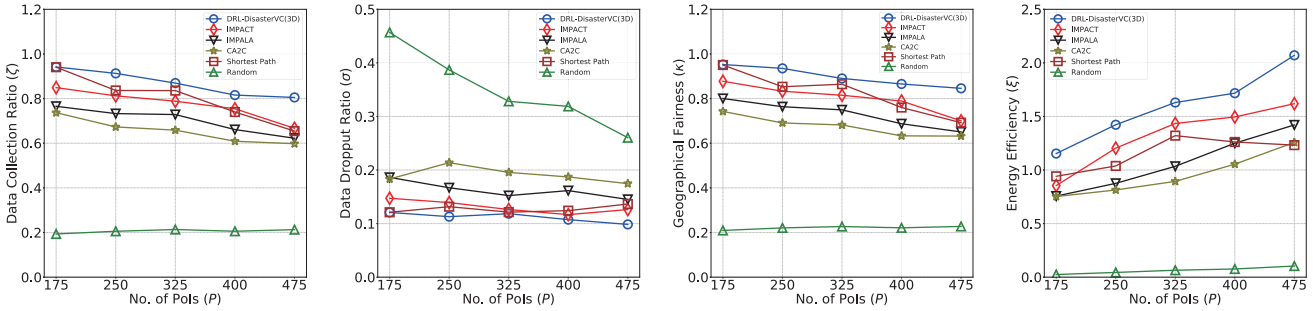
Table 2: Impact of two hyperparameters.

		$H = 1$	$H = 2$	$H = 4$	$H = 8$	$H = 16$
$b_K = 1$	ζ	0.780	0.843	0.840	0.836	0.840
	σ	0.140	0.127	0.131	0.133	0.135
	κ	0.814	0.873	0.877	0.866	0.852
	ξ	1.117	1.275	1.309	1.186	1.201
$b_K = 2$	ζ	0.821	0.905	0.913	0.879	0.855
	σ	0.127	0.119	0.117	0.124	0.127
	κ	0.852	0.921	0.934	0.912	0.879
	ξ	1.191	1.402	1.388	1.262	1.193
$b_K = 3$	ζ	0.850	0.890	**0.920**	0.874	0.808
	σ	0.122	0.120	**0.109**	0.129	0.142
	κ	0.862	0.927	**0.943**	0.892	0.840
	ξ	1.238	1.358	**1.437**	1.235	1.135
$b_K = 4$	ζ	0.843	0.864	0.874	0.830	0.797
	σ	0.129	0.123	0.119	0.134	0.158
	κ	0.860	0.894	0.905	0.871	0.818
	ξ	1.150	1.316	1.317	1.181	1.072

6.3 Comparing with Five Baselines

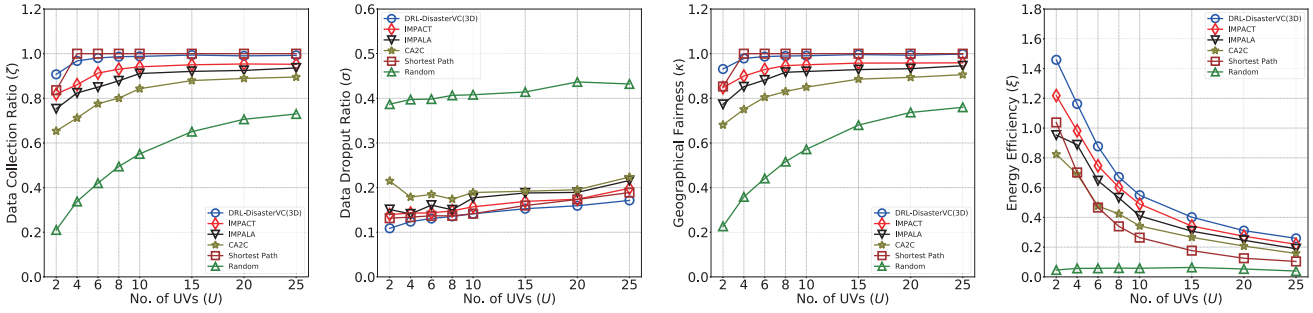
(1) **IMPALA** [2]: It is considered as the state-of-the-art asynchronous and distributed DRL approach.
(2) **IMPACT** [13]: It is an extension of IMPALA which also includes a circular buffer and a target network to improve experience sampling efficiency.
(3) **CA2C** [6]: It implements an actor-critic algorithm for multi-vehicle trajectory design which we consider as the state-of-the-art approach for VC.
(4) **Shortest Path (SP)**: Each UV finds the shortest path by genetic algorithm (GA) to visit a sequence of PoIs.
(5) **Random**: UVs choose actions $\boldsymbol{a}_t$ from $\mathcal{A}$ randomly.

6.3.1 Impact of No. of PoIs. We first vary the number of PoIs P from 175 to 475 when fixing $U = 2$, $snr_0 = 15$dbm. As shown in Figure 4, more deployed PoIs obviously impose more challenges for UVs, since the obtained higher reward at the beginning of training may lead to a local suboptimal solution. Without effective spatial explorations and relational representations delivered by DRL-DisasterVC(3D), data collection ratio ζ and geographical fairness κ of IMPACT, IMPALA, CA2C decrease significantly, but their energy efficiency ξ all rise, due to increased data density. As a result, UVs can collect more data without moving far away (i.e., lower energy consumption), thus trapped to stay at the local optimal. Even under this case, our approach still achieves $\kappa = 0.84$ and $\xi = 2.07$ when



(a) ζ (b) σ (c) κ (d) ξ

Figure 4: Impact of No. of PoIs.



(a) ζ (b) σ (c) κ (d) ξ

Figure 5: Impact of No. of UVs.

$P = 475$, which is 27%, 45%, 64%, 68% higher than IMPACT, IMPALA, CA2C and SP, respectively. Furthermore, GA based SP fails to find an optimal routing path when the solution space becomes large, e.g., from Figure 4d, we observe that the gap of ξ between SP and other algorithms gets wider with the increase of P.

6.3.2 Impact of No. of UVs. Next, we show the impact of no. of UVs by fixing $P = 250$, $snr_0 = 15$dbm while changing $U = [2, 25]$. From Figure 5, we observe that:

(1) DRL-DisasterVC(3D) consistently outperforms five baselines in terms of energy efficiency ξ in Figure 5d. For example, when $U = 4$, DRL-DisasterVC(3D) achieves a high energy efficiency of 1.16 with an 18% improvement compared to the best baseline IMPACT. Moreover, for energy efficiency, DRL-DisasterVC(3D) significantly outperforms 16%, 34%, 75%, 96% over IMPACT, IMPALA, CA2C and SP, respectively.

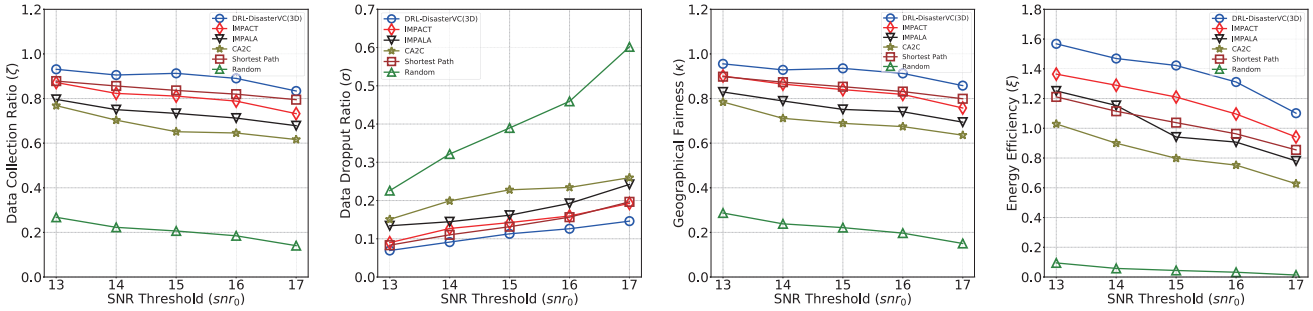
(2) From Figure 5a and Figure 5c, we can see that ζ and κ both increase when more UVs are deployed, but energy efficiency ξ drops rapidly because of the multiplied energy consumption. With the help of MHRA and PC for spatial exploration, our method enforces UVs to learn to cooperate which slows down the decreasing trend of ξ. This is why IMPACT and IMPALA perform worse than DRL-DisasterVC(3D). As shown in Figure 5d, DRL-DisasterVC(3D) achieves $\xi = 0.54$ when 10 UVs are deployed while IMPACT gets only 0.48. On the other hand, the inefficient utilization of previous experiences in IMPALA makes it hard to control large number of UVs and may trap into local optima (whose $\xi = 0.40$, which is 34% lower than DRL-DisasterVC(3D) without the help of RER). It is same as intuition that too many UVs (e.g., $U = 25$) will not bring further benefit as shown in Figure 5d.

(3) We can see that the SP nearly collect all data when deploying 4 or more UVs but its energy efficiency only reaches 0.70 maximally. This is because SP only finds the shortest path without considering energy efficiency by multi-UV collaboration, e.g., the result shows that when $U = 6$, the energy consumption of DRL-DisasterVC(3D) and SP are 2455.82kJ and 4740.46kJ, respectively. Since we introduced MHRA, the geographical relationship between different areas is fully exploited thus there is no need for UVs to visit every PoI sequentially like SP.

6.3.3 Impact of SNR Threshold. Finally, we show the impact of required data quality (reflected by received SNR threshold) in Figure 6. We fix $U = 2$, $P = 250$, and changing snr_0 from 13dbm to 17dbm. The result shows that:

(1) DRL-DisasterVC(3D) achieves the best energy efficiency, and improves 16%, 38%, 68%, 32% over IMPACT, IMPALA, CA2C and SP methods of all SNR thresholds, respectively.

(2) High snr_0 drives data collection ratio ζ to decrease and data dropout ratio σ to increase of all six algorithms monotonically, since this leads to the smaller amount of PoIs to successfully upload their data to a UV by the tx rate constraint. From Figure 6, we can see ζ and ξ of IMPACT and IMPALA drop rapidly when $snr_0 > 15$dbm due to the lack of spatial exploration. DRL-DisasterVC(3D) suffers less thanks to the help of PC, that when $snr_0 = 17$dbm, it still achieves $\zeta = 0.83$ while IMPACT and IMPALA only attain 0.73 and 0.67, respectively. CA2C collects even fewer due to its insufficient training experience collection by classical actor-critic architecture without asynchronous multi-actor-one-learner framework. Meanwhile, SP is not affected much when snr_0 increases, but it maintains poor energy utilization compared with DRL-DisasterVC(3D).



(a) ζ (b) σ (c) κ (d) ξ

Figure 6: Impact of SNR threshold.

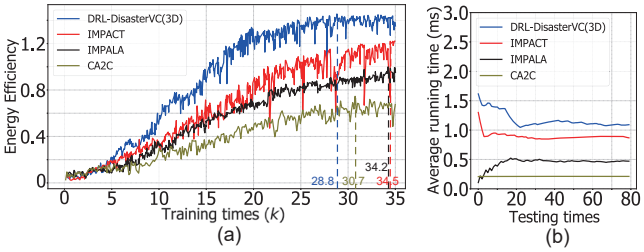


Figure 7: Computational complexity analysis.

6.3.4 Complexity Analysis. We show the computational complexity of four DRL methods in Figure 7a. We see that DRL-DisasterVC(3D) converges faster than IMPACT, IMPALA and CA2C, due to the improved sample efficiency brought by RER and learning stability by PC and MHRA. Its improved network structure does not bring extra overhead, as confirmed in Figure 7b that running time to produce actions for all UVs in a timeslot by DRL-DisasterVC(3D) is almost identical as IMPACT and only slightly higher than IMPALA and CA2C (but with scale of millisecond, which is negligible in practice).

7 CONCLUSION

In this paper, we proposed DRL-DisasterVC(3D), a distributed DRL framework with RER and clipped target network (for learning efficiency and stability improvement), and a 3D-CNN with MHRA and auxiliary PC (for spatial exploration). We design DisasterSim, a novel disaster response simulator and visualize the trajectories of UVs. Finally, we compare DRL-DisasterVC(3D) with five baselines and results show that DRL-DisasterVC(3D) outperforms all others in terms of four metrics, including data collection ratio, data dropout ratio, geographical fairness and energy efficiency.

8 ACKNOWLEDGMENTS

This work has been supported in part by the National Key Research and Development Plan of China (Grant No. 2018YFB1003701), in part by the National Natural Science Foundation of China (Grant No. 62022017). Corresponding author: Chi Harold Liu.

REFERENCES

[1] Yue Cui, Liwei Deng, Yan Zhao, Bin Yao, Vincent W. Zheng, and Kai Zheng. 2019. Hidden POI Ranking with Spatial Crowdsourcing. In *KDD'19*. 814–824.
[2] Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, et al. 2018. IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures. In *ICML'18*, Vol. 80. 1406–1415.
[3] Seungyul Han and Youngchul Sung. 2019. Dimension-wise importance sampling weight clipping for sample-efficient reinforcement learning. In *ICML'19*. 2586–2595.
[4] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, et al. 2017. Emergence of Locomotion Behaviours in Rich Environments. *CoRR* abs/1707.02286 (2017).
[5] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, et al. 2018. Rainbow: Combining Improvements in Deep Reinforcement Learning. In *AAAI'18*. 3215–3222.
[6] Jingzhi Hu, Hongliang Zhang, Lingyang Song, Robert Schober, and H Vincent Poor. 2020. Cooperative internet of UAVs: Distributed trajectory design by multi-agent deep reinforcement learning. *IEEE Transaction on Communications* 68, 11 (2020), 6807–6821.
[7] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z. Leibo, et al. 2017. Reinforcement Learning with Unsupervised Auxiliary Tasks. In *ICLR'17*.
[8] Rajendra K Jain, Dah-Ming W Chiu, William R Hawe, et al. 1984. A quantitative measure of fairness and discrimination. *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA* (1984).
[9] Boyang Li, Yurong Cheng, Ye Yuan, Guoren Wang, and Lei Chen. 2019. Three-Dimensional Stable Matching Problem for Spatial Crowdsourcing Platforms. In *KDD'19*. 1643–1653.
[10] Chi Harold Liu, Zheyu Chen, and Yufeng Zhan. 2019. Energy-Efficient Distributed Mobile Crowd Sensing: A Deep Learning Approach. *IEEE Journal on Selected Areas in Communications* 37, 6 (2019), 1262–1276.
[11] Chi Harold Liu, Yinuo Zhao, Zipeng Dai, Ye Yuan, Guoren Wang, et al. 2020. Curiosity-Driven Energy-Efficient Worker Scheduling in Vehicular Crowdsourcing: A Deep Reinforcement Learning Approach. In *IEEE ICDE'20*. 25–36.
[12] Jia-Xu Liu and Ke Xu. 2020. Budget-aware online task assignment in spatial crowdsourcing. *World Wide Web* 23 (01 2020), 289–311.
[13] Michael Luo, Jiahao Yao, Richard Liaw, Eric Liang, and Ion Stoica. 2020. IMPACT: Importance Weighted Asynchronous Architectures with Clipped Target Networks. In *ICLR'20*.
[14] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, et al. 2016. Asynchronous Methods for Deep Reinforcement Learning. In *ICML'16*. 1928–1937.
[15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, et al. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.
[16] Wangze Ni, Peng Cheng, Lei Chen, and Xuemin Lin. 2020. Task Allocation in Dependency-aware Spatial Crowdsourcing. In *IEEE ICDE'20*. 985–996.
[17] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2016. High-Dimensional Continuous Control Using Generalized Advantage Estimation. In *ICLR'16*.
[18] Qian Tao, Yongxin Tong, Zimu Zhou, Yexuan Shi, Lei Chen, and Ke Xu. 2020. Differentially Private Online Task Assignment in Spatial Crowdsourcing: A Tree-based Approach. In *IEEE ICDE'20*. 517–528.
[19] Yongxin Tong, Lei Chen, and Cyrus Shahabi. 2017. Spatial Crowdsourcing: Challenges, Techniques, and Applications. *Proc. VLDB Endow.* 10, 12 (Aug. 2017), 1988–1991.
[20] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, et al. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* 575, 7782 (2019), 350–354.
[21] Detian Zhang, Shiting Wen, Fei Chen, Zhixu Li, and Lei Zhao. 2020. Spatial crowdsourcing based on Web mapping services. *World Wide Web* 23 (01 2020).
[22] Haitao Zhao, Haijun Wang, Weiyu Wu, and Jibo Wei. 2018. Deployment Algorithms for UAV Airborne Networks Toward On-Demand Coverage. *IEEE Journal on Selected Areas in Communications* 36, 9 (2018), 2015–2031.