

Off Policy or On Policy? Multi-Agent Reinforcement Learning for Drone Swarm Coordination

Alex Shaffer, Jett Carruth
Stanford University
450 Jane Stanford Way, Stanford, CA

alexshaf@stanford.edu, jcarruth@stanford.edu

Abstract

Applications of multi-agent reinforcement learning remain underexplored. One promising application of multi-agent reinforcement learning is multi-drone coordination with potential health and military applications. In this paper, we train swarms of four drones to reach a single fixed target without colliding, using multi-agent PPO (MAPPO), and multi-agent SAC (MASAC) to compare the effectiveness of on policy and off policy learning in the multi-agent drone swarming context. Both algorithms are implemented used a Centralized Training Decentralized Execution (CTDE) framework where critic networks are trained on the global state of all drones and actor networks are trained on the local observations of each drone. The pybullet-gym-drone environment is used for all simulation physics. MAPPO is implemented with a single global value critic and MASAC is implemented with a global Q-critic; these global critics assign rewards based on the global state of all agents with no individual drone specific reward attribution at a given timestep. Local actor networks parameterize action distributions given a drone’s actors local observation. Using both MAPPO and MASAC, we successfully train drone agents to reach a target and then fly within the target perimeter while avoiding collisions under multiple local observation frameworks: (1) a drone’s local observation consists of only its kinematics (i.e. blind to other drones), (2) a drone’s local observation consists of only its kinematics and the relative kinematics of its neighbor, and (3) a drone’s local observation consists of its kinematics and the relative kinematics of all its neighbors. Thus, this paper establishes that in the context of a fixed-target, collision avoidance, drone swarming task, individual reward assignment is not necessary for learning and group-level, global state reward assignment is sufficient for both MAPPO and MASAC.

1. Introduction

Small cheap drones are becoming increasingly prevalent in military applications [8]. They also have underexplored applications in medicine and transport where they could be used transport critical medical supplies to difficult to reach areas [5]. Training drone agents could relieve the critical bottleneck of a pilot in these applications. Furthermore, having drone agents that can operate in swarms would further extend the utility of cheap drone agents for carrying out critical tasks. To this end, this paper investigates the potential application of multi-agent reinforcement learning to drone swarm coordination problems; it examines the trade-offs between off-policy and on-policy learning for training agents to coordinate in swarms, comparing MAPPO and MASAC without BC pre-training in the simulated environment of pybullet-gym-drone.

Existing work has been done showing that swarms of drone agents trained using MAPPO for formulation flying tasks can learn policies that zero-shot generalize to real world flying [1]. Such work demonstrates the utility of simulation learning for drone coordination tasks. This paper studies drone swarm coordination for a slightly different task: a reach and hover task for a single target. Rather than using a reward function to learn a specific formation, no formation is enforced. Drone agents instead need to reach a single target without colliding and then maintain close proximity to that target for the rest of the episode. While we initially planned to use MAPPO and MASAC to train agents for a suite of tasks (e.g. two targets, four targets, search and find for multiple targets), we focus on the task of finding a single-fixed target for a number of reasons. First, even the simple challenge of having four drones navigate roughly a meter each proved exceptionally difficult, especially without BC pre-training. Second, by not maintaining multiple targets we can better study collision avoidance behavior. With a single target, drones must enter a much more confined region to receive high rewards allowing the study of whether collision free episodes can emerge through the

use of proximity penalties and coverage bonuses. In the multi-target setting, drones should spread out as they go to different targets, making collisions less of a problem.

Furthermore, this paper contributes to the existing multi-agent reinforcement learning literature. Past work on MAPPO has shown individual agent reward attribution is not necessary for learning to occur in multi-agent settings; in particular, reward attribution based on a global state that includes all agents is sufficient for learning good policies in cooperative settings [12]. In both MAPPO and MASAC, we provide no individual reward attribution, only assigning rewards for the global state of all four drones. Thus, we investigate whether this setup which assigns the same rewards to all drones at a single timestep is conducive to learning for four homogeneous drone agents in cooperative setting. Furthermore, this paper builds on work done in applying off-policy learning for multi-agent reinforcement learning. Combining aspects of MADDPG [6], single SAC [3], and previous multi-agent SAC implementations [10], we implement MASAC for four agent drone coordination with a single global critic and no individual agent reward attribution. Both MAPPO and MASAC implementations are designed with scaling the number of agents in mind; since we use a global critic and implement actor networks for local drone observations with parameters, the number of networks needed for $n > 4$ will remain the same as in the four drone case experimented with in this paper (although depending on the observation mode more network parameters might be required for successful learning).

Since actor networks rely on local drone observations and critic networks rely on global state observations under the CTDE framework used, we test a number of local drone observations for the actor network meant to simulate real-world constraints on inter-drone communication. First, we test a local drone observation in which the acting drone has no information about the other drones and only has access to its kinematic information (i.e. flying blind). Second, we test a local drone observation in which the acting drone only has information its kinematics and relative kinematics of its nearest neighbor. Finally, we test when the local observation has complete relative kinematic information about all four drones in the swarm. Together, this helps to position this paper as a step toward future multi-agent drone swarm RL research, particularly given the ultimate finding that even without BC pre-training to initialize actor networks, swarms learned successful policies using both MAPPO and MASAC.

2. Methods

In the methods section, we outline the two implementations of MAPPO tested, the implementation of MASAC tested, reward engineering, and the observation ablation for local actor networks. All implemen-

tations are tested on a consistent task: the four drone agents are initialized hovering at positions $(0.15\text{m}, 0.15\text{m}, 0.5\text{m})$, $(0.15\text{m}, -0.15\text{m}, 0.5\text{m})$, $(-0.15\text{m}, -0.15\text{m}, 0.5\text{m})$ and $(-0.15\text{m}, 0.15\text{m}, 0.5\text{m})$. They need to navigate within a 0.5m radius of a target at $(1\text{m}, 1\text{m}, 1.5\text{m})$ i.e. they must navigate up and to the right. Episodes are always 500 steps with a control frequency of 30 Hz. Hence, each episode is 16.6 seconds. Episodes are always truncated instead of having a termination condition since assigning negative rewards in a setup with a termination could result in a policy in which drones crash immediately to avoid receiving negative rewards over a longer episode. Thus, to simplify the problem setup and avoid having to engineer a properly scaled termination reward for something like a crash event, we do not include a termination condition with episodes always truncated.

The desired behavior for the drone agents is to navigate to the target and hover within a 0.5 meter radius of it until the end of the episode without colliding with other agents in the swarm. Positive rewards are assigned for being near the target and negative rewards are assigned when drones get within too close of proximity of each other (see for 2.3 for specific reward function details). During roll out, at each episode step, each drone acts according to the actor network which takes in its local observation; since agents are homogeneous and thus we employ parameter sharing for the actor networks, all agents use the same actor network to act. Both MAPPO and MASAC actor networks parametrize a continuous distribution over actions. At evaluation time, drone actors take the mean action from these parametrized distributions. During roll outs, evaluations occur every 50,000 steps and consist of ten 500 step episode roll outs i.e. approximately 166 seconds on experience.

2.1. MAPPO

The implementation for MAPPO is based on **Yu et al. 2021** [11]. The critic network $V_{\phi}^{\pi}(s_{\text{global}})$ is a function of the global state of all four drone agents' kinematic information and target information; hence,

$$s_{\text{global}} = (\text{drone \#1 absolute kinematic state,} \\ \text{drone \#2 absolute kinematic state,} \\ \text{drone \#3 absolute kinematic state,} \\ \text{drone \#4 absolute kinematic state,} \\ \text{target-reached flag,} \\ \text{absolute target position})$$

The actor network implemented using parameter sharing only takes in the i^{th} drone's observation o_t^i at timestep for which we test a number of setups (see 2.4). The actor network parameterizes a truncated normal distribution which outputs actions $[-1, 1]$ for each dimension of the four dimensional velocity action vector. The choice to use a sin-

single actor network for all drones that is a function of a single drone agent’s local observation is motivated on previous MAPPO literature which suggests this parameter sharing approach is effective for homogeneous agents [12]. By only using a single actor network, this paper tests an approach that would potentially scale with more than four drones since separate networks would not need to be trained for all of them.

To train the agents using MAPPO, we collect roll outs of the current policy. Advantage is computed using the global state of all drones and a GAE advantage estimation,

$$\begin{aligned} \delta_t &= r_t + \gamma V_\phi(s_{t+1}^{\text{global}}) - V_\phi(s_t^{\text{global}}) \\ \hat{A}_t &= \delta_t + \gamma \lambda \hat{A}_{t+1}. \end{aligned}$$

We then compute clip loss for the actor network using a drone agent’s local observation and sampling from the roll out buffer with observations from the current policy; thus, for a single local observation from drone i the minimization objective becomes

$$\mathcal{L}^{\text{clip}} = -\min\left(\rho_t^i \hat{A}_t, \text{clip}\left(\rho_t^i, 1 - \epsilon, 1 + \epsilon\right) \hat{A}_t\right)$$

where

$$\rho_t^i = \frac{\phi_\theta(a_t^i | o_t^i)}{\phi_{\theta_{\text{old}}}(a_t^i | o_t^i)}.$$

Notice the clip loss objective is computed conditional on drone i ’s local observation o_t^i and there is no individual reward signal for agent i since advantage is a function global state. Similar to single agent PPO, the inclusion of the clip loss allows for multiple gradient steps each policy update via importance sampling. Summing over local observations sampled from all agents, the loss becomes

$$\mathcal{L}^{\text{clip}} = -\frac{1}{B} \sum_{(i,t)} \min\left(\rho_t^i \hat{A}_t, \text{clip}\left(\rho_t^i, 1 - \epsilon, 1 + \epsilon\right) \hat{A}_t\right)$$

with ρ_t^i defined as before. Using the global state, one can then fit the value function $V_\phi(s_t^{\text{global}})$ under the new policy using mean-squared error loss where returns are computed as the sum of global state advantages and values. Finally, entropy minimization objective can be computed as

$$\mathcal{L}^{\text{entropy}} = \frac{1}{B} \sum_{(i,t)} \log(\pi_\theta(a_t^i | o_t^i))$$

so that entropy is maximized, incentivizing agent exploration. Thus, the total loss minimization objective becomes

$$\mathcal{L}^{\text{clip}} + c_{\text{value}} \mathcal{L}^{\text{value}} + c_{\text{entropy}} \mathcal{L}^{\text{entropy}}$$

similar to the single agent PPO setting.

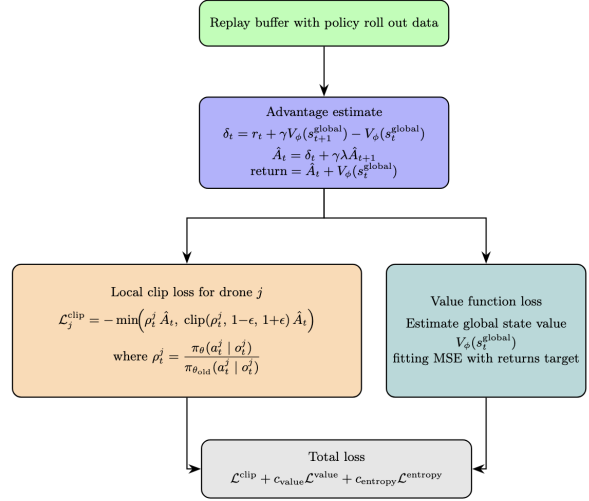


Figure 1. Calculating MAPPO minimization objective. Loss is fit to pick actions that maximize global state advantage, fit loss function, and maximize entropy.

2.1.1 MAPPO without Attention

For this paper’s base MAPPO implementation, we use a three layer MLP with ReLU parameterizing a truncated normal for the actor network. Similarly, a three layer MLP with layer normalization is used for the global value network [9]. We use orthogonal initialization for all linear layers.

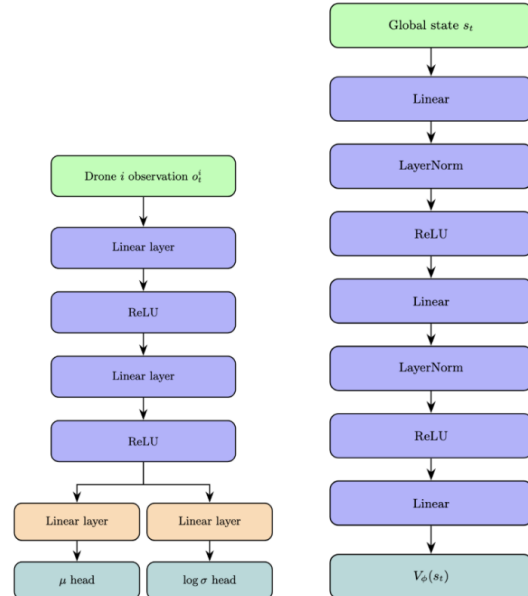


Figure 2. Actor network (left). Critic network (right). Under the CTDE framework actor network takes in local observation of a drone while global value network takes in global observation of all four drones.

2.1.2 MAPPO with Attention

In addition to the base MAPPO implementation, we also implement MAPPO with a single-headed self-attention layer. This tokenizes the absolute kinematics of all four drones in the critic. In the actor, the absolute kinematics of the acting drone is tokenized and the relative kinematics of the other drones are tokenized. An additional token giving the relative position of the target to the actor drone is provided. Linear layers are used to project all tokens into the same dimension. Since attention is permutation invariant, we add a positional encoding to tokens for the actor network since the tokens have different meanings [2]. However, for the global critic we use no positional encoding since the same positional layout of drones should have the same value regardless of how the drone numberings are permuted (i.e. if the positions and velocities of drones are assigned to specific values it should not matter if drone #1 and drone #2 are swapped). This implementation is only tested with local observation all setup (see 2.3) since the actor network requires the drone agent to have relative kinematic information about the other three drones. We train MAPPO with an attention network with future research in mind; specifically, in future research with n instead of 4 drones, we envision a MAPPO actor that has an attention layer over its k nearest neighbors with $k \leq n$.

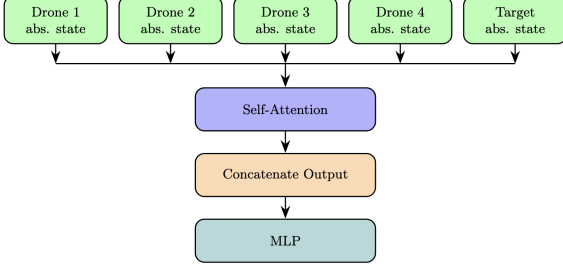


Figure 3. Critic network with attention where MLP returns $V_{\phi}(s_t^{\text{global}})$.

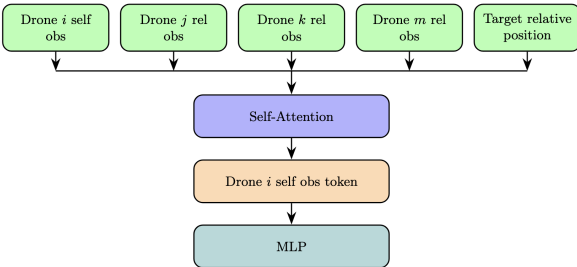


Figure 4. Actor network with self-attention for drone i . Allows the relative kinematics of drones other than acting drone i to attend to drone i kinematics token. MLP output parameterizes truncated normal distribution as before.

2.2. MASAC

Unlike MAPPO, we only provide a single implementation for MASAC with MLPs for actor and critic networks similar to Figure 2. Like **Lowe et al. 2017**'s implementation of MADDPG this paper uses critic networks conditioned on global state information of all drones [6]. However, unlike this previous implementation we do not maintain n critic networks for each agent, instead choosing to use a global critic network that fits $Q^{\pi}(s_t^{\text{global}}, a_t^1, a_t^2, a_t^3, a_t^4)$ i.e. a global Q -critic over the joint action of all drones. Furthermore, this paper's implementation does not attribute rewards individually to each agent, but rather uses a single reward attribution for the global state (as in the MAPPO setup). The reasons for these design choices are two fold. First, some aspects of single agent SAC [3] are borrowed that would not scale well with critic networks for each actor. We use two Q -critic estimates $Q_{\phi_1}^{\pi}$ and $Q_{\phi_2}^{\pi}$ with target networks $Q_{\phi_1^{\text{target}}}^{\pi}$ and $Q_{\phi_2^{\text{target}}}^{\pi}$ to mitigate the impact of Q -overestimation. Furthermore, borrowing on SAC, we maintain an entropy term with coefficient α . Thus, similar to the single agent SAC, the TD target for Q -estimation becomes,

$$\begin{aligned} \text{TD Target} &= r + \gamma \min_{j=1,2} Q_{\phi_j}(s_{t+1}^{\text{global}}, a_{t+1}^1, a_{t+1}^2, a_{t+1}^3, a_{t+1}^4) \\ &\quad - \alpha \log \pi(a_{t+1}^1, a_{t+1}^2, a_{t+1}^3, a_{t+1}^4 | s_{t+1}^{\text{global}}) \\ &= r + \gamma \min_{j=1,2} Q_{\phi_j}(s_{t+1}^{\text{global}}, a_{t+1}^1, a_{t+1}^2, a_{t+1}^3, a_{t+1}^4) \\ &\quad - \alpha \log \pi(a_{t+1}^1, a_{t+1}^2, a_{t+1}^3, a_{t+1}^4 | o_{t+1}^1, o_{t+1}^2, o_{t+1}^3, o_{t+1}^4) \\ &\approx r + \gamma \min_{j=1,2} Q_{\phi_j}(s_{t+1}^{\text{global}}, a_{t+1}^1, a_{t+1}^2, a_{t+1}^3, a_{t+1}^4) \\ &\quad - \alpha \sum_i \log \pi(a_{t+1}^i | o_{t+1}^i). \end{aligned}$$

where $a_{t+1}^1, \dots, a_{t+1}^4$ are sampled using the local actor network given $o_{t+1}^1, \dots, o_{t+1}^4$, respectively. The inclusion of the SAC-style entropy term in the multi-agent setting causes some problems not present in the single agent setting. Since, the actor network is a function of a drone's local observation, we approximate

$$\log \pi(a_{t+1}^1, a_{t+1}^2, a_{t+1}^3, a_{t+1}^4 | s_{t+1}^{\text{global}}) \approx \sum_i \log \pi(a_{t+1}^i | o_{t+1}^i).$$

Finally, for simplicity we do not treat α as learnable as later SAC implementations do [4], instead treating it as a hyperparameter as in the original 2018 SAC paper [3].

The actor network takes in a drone's local observation and uses a three layer MLP with mean and log standard deviation heads to parameterize a normal distribution. To allow gradients to propagate through it we use the reparameterization trick; to ensure actions are in $[-1, 1]$ outputs from the distribution are squashed with tanh such that

$a_t = \tanh(u_t)$ where u_t is sample from the Gaussian parameterized by the actor network with log probabilities calculated based on this squashed distribution [3]. Thus the maximization objective for the actor network becomes

$$Q_{\phi_1}(s_t^{\text{global}}, a_t^1, a_t^2, a_t^3, a_t^4) - \alpha \log \pi(a_t^i | o_t^i)$$

for a single local observation from drone i . o_t^i and s_t^i are sampled from the replay buffer. a_t^1, \dots, a_t^4 are sampled conditional on o_t^1, \dots, o_t^4 . However, a_t^j for $j \neq i$ are treated as fixed and gradients are only backpropagated through a_t^i . Notice the loss function is nearly identical to the single agent SAC version; just adapted to the multi-agent setting where the global critic operates over global states and the joint actions of all agents. Loss is calculated over batches sampled from the replay buffer and the actor network only relies on the local observation o_t^i of a single agent during test time.

2.3. Reward Engineering

To get drone agents to reach the target without colliding and then stay near it substantial reward engineering is necessary. All reward functions tested are some function of a collision or proximity penalty with positive rewards for being near the target. The final reward function we settle on for ablations consists of the following components. First, a distance penalty which is the negative sum of distance of all drones from the target of all drones outside the target perimeter. Second, a proximity penalty of the negative sum of the number of pairs of drones less than 0.1 meters apart is used to penalize likely collision behavior. Third, a reward of +1 is given for each drone inside the target perimeter at a given episode step. Finally, a coverage bonus of the pairwise sum of distances of the drone agents is given. This coverage bonus is capped at +3 since we do not want to prioritize agents spreading out instead of reaching the target; with a cap of +3 the max coverage bonus can be achieved by the swarm with all agents within 0.5 meters of the target (i.e. within the target perimeter). These bonuses and penalties are then summed to yield the reward for a given timestep. For an untrained swarm of four drones it yields average episode rewards of approx. -2000 and for a trained swarm it yields average episode rewards of approx. +2000. Since negative rewards are assigned, episodes are never terminated as we worry termination could result in reward hacking behavior where agents crash immediately. While reward engineering is not the focus of this paper; it was necessary to experiment in order to get a reward function that incentivizes the desired behavior. In future, more substantial reward engineering would likely result in behavior more in line with the desired task.

2.4. Local Observation Ablation

For both MAPPO and MASAC the actor network takes in a local observation o_t^i and outputs mean and standard deviation to parameterize a distribution over actions. For MAPPO without attention and for MASAC (for which we provide no implementation with attention) this paper performs an ablation over what the vector o_t^i contains, meant to test different possible communication frameworks between drones at test time.

One, a self-observation framework is tested. In this setup,

$$o_t^i = (\text{drone}_i \text{ kinematic state,} \\ \text{drone}_i \text{ relative position to target})$$

where kinematic state includes drone i 's past 15 actions. This is meant to simulate a setup in which drone agents have no inter-drone communication but still need to coordination to the target without colliding.

Two, a nearest-neighbor observation framework is tested. In this setup,

$$o_t^i = (\text{drone}_i \text{ kinematic state,} \\ \text{drone}_j \text{ relative kinematic state,} \\ \text{drone}_i \text{ relative position to target})$$

where drone j is drone i 's nearest neighbor. This is meant to simulate a setup in which drone i only has access to relative information about its nearest neighbor during test time.

Three, an all-observation framework is tested. In this setup,

$$o_t^i = (\text{drone}_i \text{ kinematic state,} \\ \text{drone}_j \text{ relative kinematic state,} \\ \text{drone}_k \text{ relative kinematic state,} \\ \text{drone}_m \text{ relative kinematic state,} \\ \text{drone}_i \text{ relative position to target})$$

where drones j, k, m are the other three drones. This is the only observation framework we test out MAPPO with attention implementation for.

This local observation ablation helps generalize results to real-world use cases. Real-world scenarios may have constraints on what a drone's local observation contains that do not exist in the pybullet-gym-drone environment. Thus comparing MAPPO and MASAC policy performance across different communication frameworks ranging from no to communication (only a drone's local state is known to the actor network) to complete communication (the kinematics of a drone and all relative kinematics of all other drones are known to the actor network) are critical to having generalizable results.

3. Results

In both, MAPPO and MASAC the learned drone policy across local observation modes, is able coordinate all for drones to the target. There are some notable differences between MAPPO and MASAC however. MAPPO has far more variable performance across seeds. In seeds for which learning occurs, it is comparable to MASAC with all drones reaching the target without collisions. However, in MAPPO, on at least one seed across observation modes, no learning will occur and the policy will exhibit a behavior where all drones immediately crash. The MAPPO policies for self-observation mode where the actor network only has access to drone kinematics of acting drone exhibit surprising behavior for seeds in which learning occurs. Specifically, for both the two seeds from MAPPO in which learning occurs for actor self-observation only mode, policies are learned which have 0 collisions during all 10 evaluation episodes, with all four drones reaching the target. MASAC with self-observation mode on the other hand never learns a policy which has 0 collisions and all four drones reaching the target. We explore algorithm specific results in the following subsections in more depth.

Method	Reward	# Collisions	Num. Drones Reach Target
MAPPO (Self)	1277 ± 632	1.5 ± 1.3	2.6 ± 1.1
MAPPO (Nearest Neighbor)	846 ± 700	0 ± 0	2.6 ± 1.1
MAPPO (All Agents)	-778 ± 49	0 ± 0	0 ± 0
MAPPO + Attention (All Agents)	744 ± 1003	0 ± 0	2.6 ± 1.1
MASAC (Self)	2763 ± 66	7.7 ± 1.5	4 ± 0
MASAC (Nearest Neighbor)	3225 ± 77	0 ± 0	4 ± 0
MASAC (All Agents)	3193 ± 5	0 ± 0	4 ± 0

Table 1. Avg. performance ± standard error across seeds during best evaluation run (10 episodes, 500 steps each). Notice MASAC more consistent. MAPPO on performant seeds is comparable to MASAC with all drones reaching target. Collision measured as any violation of < 0.1m proximity.

3.1. MAPPO without Attention Results

MAPPO performance was highly seed dependent; when learning occurred, the performance of the final policy was comparable to MASAC. MAPPO performed exceptionally well in the self-observation mode for the actor network. Despite each drone only having information about its local kinematic information at test time, across two of three seeds in the final evaluation run, the actor network learned a policy which was able to completely avoid collisions (< 0.1m apart at given step) during evaluation runs of 10 episodes. We found this result surprising given in the self-observation mode setup for the actor network drones are effectively flying blind, yet they still learn an effective policy in line with the desired behavior. In the nearest neighbor observation mode setup, two of three also had performant policies with all drones reaching the target and maintain 0 collisions. Surprisingly, without attention, with the

all-observation mode, the drone agents never reached the target across seeds. Overall, these MAPPO results support Yu et al. 2021’s [12] finding that successful policies can be learned in cooperative multi-agent settings with just a global value critic and no individual reward attribution for agents.



Figure 5. MAPPO without attention final training and evaluation runs with local observation mode self for 2/3 seeds where learning occurred. Multiple eval runs during which all drones reach target without collision (< 0.1m between drones) across all 10 episodes eval (e.g. step 800,000 eval orange, step 1.4 million eval blue). See Appendix 4 for all average reward curves.

3.2. MAPPO with Attention Results

While the self-attention implementation of MAPPO could only be tested with the all-observation mode for the actor network since all relative kinematics of the acting drone are tokenized, it significantly outperformed MAPPO without attention on all-observation mode. MAPPO with attention quickly learned policies in which the swarm converged and the target and stayed near it without colliding. On the best evaluation run (10 episodes, 500 steps each), MAPPO with attention learned a policy in which all drones spent an average of over 1200 steps in the target perimeter (where we count the number of steps each drone spends in the target area i.e. max of 2000) while maintaining 0 collisions. This indicates it is possible to learn a swarm policy with MAPPO in which no agents collide and multiple agents remain in the target perimeter. Furthermore, MAPPO with attention demonstrates clearly the problem of learning to crash. Examining the episode rewards we notice two policies in which drones successfully learn to find



Figure 6. MAPPO with attention final training and evaluation runs with local observation mode all. Demonstrates that under learned policy drones find target and do not collide. The top graph gives the sum across drones of episode steps within target perimeter (out of 2000 since 4 drones \times 500 times steps total). Collisions count all violations of < 0.1 meters between drones.

the target; the other, seed 1, fails to learn anything. Examining the evaluation graphs (see Fig. 7) we notice drones are converging to a policy of immediately crashing with all of them on the ground for the majority of the episode. We hypothesize this crash convergent policy is particularly pernicious because once convergence occurs the rollout buffer is almost entirely filled with episodes in which a crash occurs. This provides minimal reward signal to learn from and inescapable performance collapse ensues. Furthermore, since parameters are shared across actors (i.e. only one actor despite taking in local observation of single drone), once this undesirable policy starts occurring for a single drone across local observation states, it spreads to the other agents.

3.3. MASAC

MASAC exhibits excellent performance with all four drones reaching the target across all observation modes. However, performant MAPPO seeds for self-observation mode converge to more desirable behavior, successfully avoiding collisions. In the self-observation mode, MASAC fails to ever reach a policy in which all drones reach the target with 0 collisions. MASAC more consistently reaches the target than MAPPO across seeds. However, MASAC proved exceptionally finicky with persistent Q -loss explosions and reward-hacking behavior.

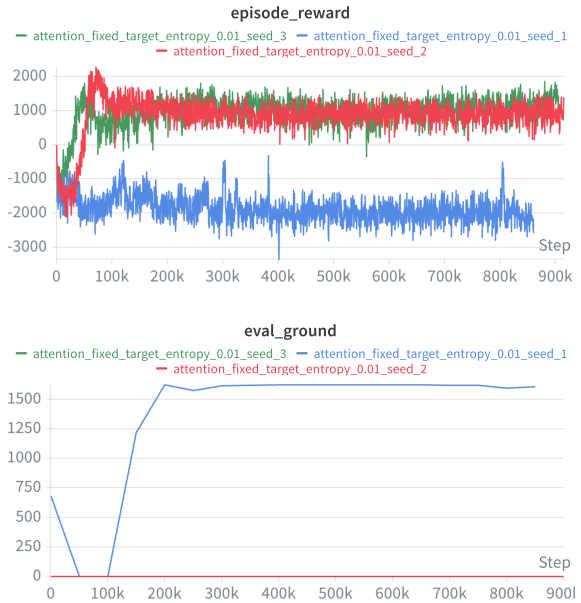


Figure 7. MAPPO with attention final training and evaluation runs with local observation mode all. Demonstrates divergent behavior across seeds. Seed 1 (blue) results in crashing behavior that is irrecoverable and no learning occurs. Bottom chart shows number of average number of time steps drones are on the ground during evaluation runs (out of 2000 since 4 drones \times 500 times steps total).

3.3.1 Reward Scaling

The original implementation of SAC found reward scaling to be a particularly difficult problem in the single agent problem setup [3]. In the multi-agent drone setup, reward scaling caused a number of problems as well. With too small of reward scaling, learning failed to occur (see Appendix 4). We also dealt with persistent Q -loss explosions across observation modes, relying on gradient clipping and $\tau = 0.005$ for the target network update to keep Q -loss as stable as possible. Treating α as a fixed hyperparameter likely did not help. In future, learning α as later implementations of single SAC [4] do would likely result in better performance and less trouble with hyperparameter tuning. While MAPPO suffered from occasional performance collapse and crash behavior for some seeds, it was much better with variable reward scales and had more stable loss with similar performance to MASAC on most seeds.

3.3.2 Reward Hacking

Furthermore, in MASAC reward hacking behavior is observed that is not observed in MAPPO. MASAC often had higher rewards but also more violations of proximity of $< 0.1m$ between drones (i.e. more collision prone behavior). This is particularly apparent in the case of self-

observation mode where MASAC learns a policy that receives comparable rewards to MAPPO but never converges to a policy with no proximity $< 0.1m$ violations. In essence, the MASAC policy has drone agents spend more time in the target perimeter, receiving high rewards for this, but at the cost of being more collision prone. This reward hacking behavior also suggest a disconnect between the reward function used and desired behavior; however, specifying a reward function from scratch is exceptionally difficult and much future research could be dedicated to task-specific reward engineering in the multi-drone context.



Figure 8. MASAC final training and evaluation runs with local observation mode self. Demonstrates reward hacking behavior during eval runs. Exceptionally collision prone behavior with up to 100s of violations of $< 0.1m$ proximity between drones.

4. Conclusion

For the task of four drones reaching a single target without colliding, both MAPPO and MASAC are both able to learn policies which only cause infrequent collisions for the drone swarms with all agents in the drone swarm reaching the target across a variety of actor local drone observation modes. MASAC demonstrates more consistent performance across seeds than MAPPO but only with MAPPO are all drones able to reach the target without collision during evaluation time when local observations for each drone only consistent of their kinematics (i.e. self observation mode). Furthermore, MASAC proves much more difficult to train than MAPPO with MASAC suffering from persistent Q -loss explosions, sensitivity to reward scaling, and reward-hacking. Over the course of research we spent approx. 300

dollars in compute training and tweaking MASAC algorithm compared to only 160 dollars in compute for MAPPO across both MAPPO implementations (with and without attention). For these reasons, we assert future research is likely better focused on MAPPO.

This paper proposes a number of avenues for future research. First, we will experiment with BC pre-training and pre-initialized policies for the actor network from the single agent setting. Initializing the actor network with a policy that already reaches the target and hovers near it will significantly cut down on training time and allow for learning more complex tasks. Then, with this pre-initialized single agent policy in the actor network we can begin training the swarm. Based on our research in which collision avoid behavior emerges through proximity penalties and coverage bonuses, we hypothesize this single agent pre-training to multi-agent MAPPO with reward shaping to penalize collisions and incentive to coverage could prove effective.

Second, drawing on implementations of MAPPO that use RNN’s in the actor and critic networks, we propose using an attention layer with positional encodings for tokenized observations (local for actor, global for critic) over the previous k time steps. We hypothesize this will solve some of that partial observability problems in MAPPO and result in better policy performance.

Finally, a natural extension of the paper would be to more drones and a wider array of tasks.

Overall, this paper demonstrates that even without individual agent reward attribution and a single global critic for joint agent state, both off policy and on policy approaches can be effective for learning drone swarm coordination with a small number of drone agents. It provides further evidence to the surprising effectiveness of MAPPO that **Yu et al. 2021 [12]** observe in cooperate settings even with the same rewards for all agents at a time step given global state instead of individual agent-specific rewards.

Team Member Contributions

All team members contributed equally to the project. Jett contributed the majority of the work to the MASAC implementation, although Alex helped. Alex contributed the majority of the work to the two MAPPO implementations, although Jett also helped. Jett and Alex split work equally on the pybullet-gym-drone environment wrapper used. Jett did the majority of the work on the poster and Alex did the majority work on the writeup. Overall we split the work such that we were both able to learn the intricacies of MAPPO and MASAC. Team member contributions roughly reflect our initial work breakdown plan.

References

- [1] S. Batra, Z. Huang, A. Petrenko, T. Kumar, A. Molchanov, and G. S. Sukhatme. Decentralized control of quadrotor

- swarms with end-to-end deep reinforcement learning. *arXiv preprint arXiv:2109.07735*, 2021. [1](#)
- [2] Fei-Fei, Adeli, Durante, Johnson, and Xiang. Cs231n lecture 8: Attention and transformer. https://cs231n.stanford.edu/slides/2026/lecture_8.pdf, 2026. [4](#)
- [3] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018. [2](#), [4](#), [5](#), [7](#), [10](#)
- [4] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2019. [4](#), [7](#)
- [5] L. Jackson. Navigating use of drones for wartime medical operations. <https://www.ausea.org/articles/navigating-use-drones-wartime-medical-operations>, 2025. [1](#)
- [6] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *arXiv preprint arXiv:1706.02275*, 2017. [2](#), [4](#), [10](#)
- [7] J. Panerati, H. Zheng, S. Zhou, J. Xu, A. Prorok, and A. P. Schoellig. gym-pybullet-drones. <https://github.com/learnsyslab/gym-pybullet-drones.git>, 2021. [10](#)
- [8] A. Roque. Under ‘drone dominance’ push, pentagon begins receiving small drones. <https://breakingdefense.com/2026/06/under-drone-dominance-push-pentagon-begins-receiving-small-drones/>, 2026. [1](#)
- [9] C. C. Staff. Cs224r spring 2026 homework 2 online reinforcement learning. https://cs224r.stanford.edu/material/hw2/CS224R_2026_Homework_2.pdf, 2026. [3](#)
- [10] Z. Woywood, J. I. Wiltfang, J. Luy, T. Enders, and M. Schiffer. Multi-agent soft actor-critic with coordinated loss for autonomous mobility-on-demand fleet control. *arXiv preprint arXiv:2404.06975*, 2024. [2](#)
- [11] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. Wu. mappo. <https://github.com/zoeyuchao/mappo.git>, 2021. [2](#)
- [12] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, and Y. Wu. The surprising effectiveness of ppo in cooperative, multi-agent games. *arXiv preprint arXiv:2103.01955*, 2021. [2](#), [3](#), [6](#), [8](#), [10](#)

One-Page Extended Abstract

Since title formatting makes it so our one-page extended abstract fits on two pages, we include it in a single page format before the Appendix.

CS 224R PSET # 2 Code Citation

We use PSET #2 as a starting point for our on policy and off policy implementations, adapting them to the the multi-agent setting. For MAPPO, we use the truncated normal

implementation for our actor network distribution and the rollout buffer.

AI Use Citation

We did not use AI for implementation code. We did ask Claude to explain the torchrl documentation for the replay buffer module which we use in our MASAC implementation. A complete transcript of the chat can be found in our final Appendix subsection (see [4](#)).

Acknowledge of Assigned Mentor Help

We thank Anushree for providing feedback on our project proposal and milestone that was helpful in scoping our project.

Extended Abstract

Applications of multi-agent reinforcement learning remain underexplored. One promising application of multi-agent reinforcement learning is multi-drone coordination with potential health and military applications. In this paper, we train swarms of four drones to reach a single fixed target without colliding, using multi-agent PPO (MAPPO), and multi-agent SAC (MASAC) to compare the effectiveness of on policy and off policy learning in the multi-agent drone swarming context. Both algorithms are implemented using a Centralized Training Decentralized Execution (CTDE) framework where critic networks are trained on the global state of all drones and actor networks are trained on the local observations of each drone. The `pybullet-gym-drone` environment is used for all simulation physics for the four `Bitcraze Crazyflie` quadcopter drone agents [7]. We train agents from scratch without behavioral cloning. All reward functions tested for both MASAC and MAPPO assign the same reward to all agents based on their global state with no individual agent-specific rewards. The action space for each drone consists of a 4-dimensional velocity vector consisting of linear velocity in each direction and yaw with each dimension normalized $[-1, 1]$; stabilization and rotor control are left to `pybullet-gym-drone`'s internal PID based on these velocity action commands. All drones start at an altitude of 0.5 meters around the origin and are required to navigate up and to the right to within 0.5 meters of a target at $(1, 1, 1.5)$ without colliding. All episodes are 500 steps with 30 control frequency Hz and no termination condition.

MAPPO is implemented based on **Yu et al. 2021** which uses a global critic $V^\pi(s_{\text{global}})$ that predicts the value of single global state for all agents [12]. Similarly, we implement local actors that take in local observations o_i from a drone agent i at a given timestep. Since all drone agents are homogeneous, implementations utilize parameter-sharing for the actor network; hence, only a single actor network is needed for all drones. Unlike **Yu et al. 2021**, we do not use an RNN in our actor and critic networks. We test two different implementations of MAPPO. In the first, actor and critic networks are three layer MLPs with layer norm in the critic network. In the second, we use attention in our actor and critic networks. In the critic network, each drone's absolute kinematic information and the target position is treated as a token and self-attention is performed before being fed into an MLP. In the actor network, the drone agent's absolute kinematic information is tokenized and the relative kinematics of the other drones and target are tokenized before a self-attention layer with the output fed into an MLP.

MASAC is implemented similarly to MAPPO with global a Q -critic and local actor networks. Unlike MAPPO, a seminal paper does not exist for MASAC. Thus, we based

our implementation on **Lowe et al. 2017**'s implementation of MADDPG and **Harnooja et al. 2018**'s implementation of single agent SAC [3] [6]. We use a global critic that estimates $Q^\pi(s_{\text{global}}, a_1, a_2, a_3, a_4)$ instead of maintaining critics for each drone conditioned on global state information. Borrowing from SAC we maintain two critic networks with respective target networks to mitigate overestimation bias with an entropy term. Finally, the actor networks is a function of a drone agent's local observation, using parameter sharing across all four drone agents' actor networks so that all drones actions need only rely on a single network. Actor and critic networks are three layer MLPs with layer norm in the critic network.

After substantial reward engineering, we perform an ablation over different local observation setups for the actor network for our MAPPO without attention implementation and MASAC implementation. We (1) train agents whose actor networks only takes in a drone's kinematic information and relative position to the target (self-observation mode), (2) train agents whose actor networks take in the kinematic information of the acting drone and the relative kinematic information of the nearest other drone (nearest-neighbor observation mode), (3) train agents whose actor networks are a function their kinematic information and the relative kinematic information of the other three drones in the swarm (all-observation mode). For MAPPO with attention we only test this final setup where the actor network takes in relative kinematic information about all other agents.

This paper finds that across all actor observation modes, we are able to train the swarm such that all four drone agents reach the target by step 160 and continue to stay within the target perimeter. In the self-observation mode, using MAPPO, the trained swarm completely avoids collisions during evaluation runs while MASAC fails to have zero collisions in the self-obs setup with > 3 per episode across evaluation runs. In the nearest neighbor and full information setup, both MASAC and MAPPO result in agent policies such that the swarm completely avoids collisions while also reaching and staying in the target perimeter during evaluation runs. We find substantially better performance of MAPPO with attention than MAPPO without attention in all-observation setup with MAPPO agents in the attention setup staying in the target perimeter for over 1300 steps (note this is bigger than 500 since we count steps across all four agents - \max of $2000 = \# \text{ agents} \times 500 \text{ timesteps}$). We document more variable learned-policies across seeds for MAPPO with agents occasionally learning to crash immediately into the ground; however, with MASAC we observe significantly more reward-hacking behavior with agents converging to policies where agents are more likely to collide if it means they can stay within the target perimeter and continue receiving rewards from this.

Appendix

Average Episode Rewards Across Algorithms and Local Drone Observation Modes

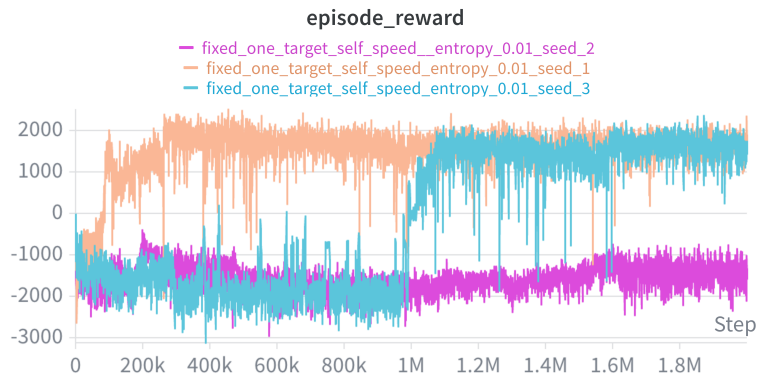


Figure 9. Average Episode Rewards for MAPPO w/o attention with actor network self-observation only.

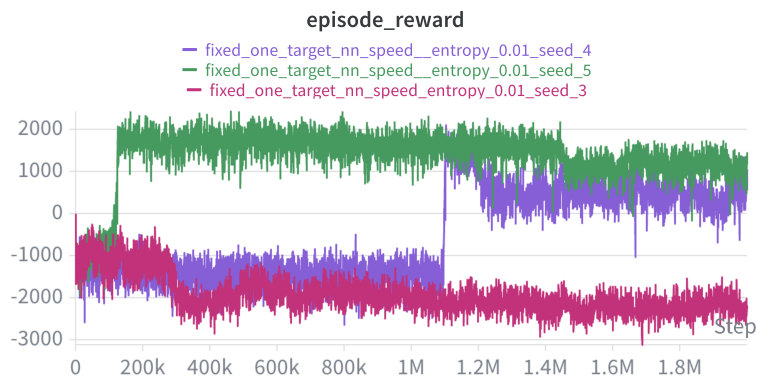


Figure 10. Average Episode Rewards for MAPPO w/o attention with actor network nearest neighbor-observation only.



Figure 11. Average Episode Rewards for MAPPO w/o attention with actor network all observation

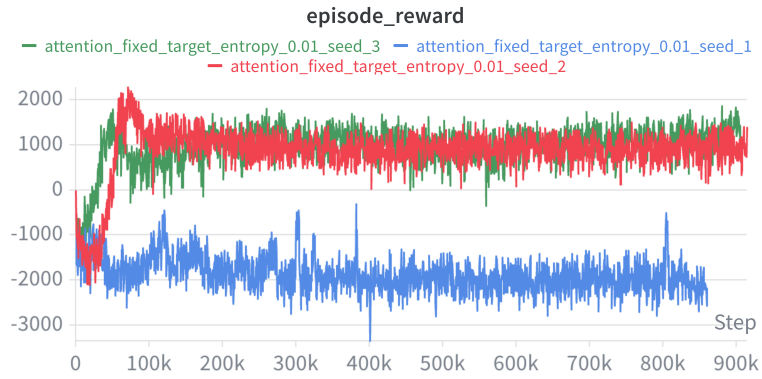


Figure 12. Average Episode Rewards for MAPPO w/ attention with actor network all observation



Figure 13. Average Episode Rewards for MASAC w/ with actor network self observation

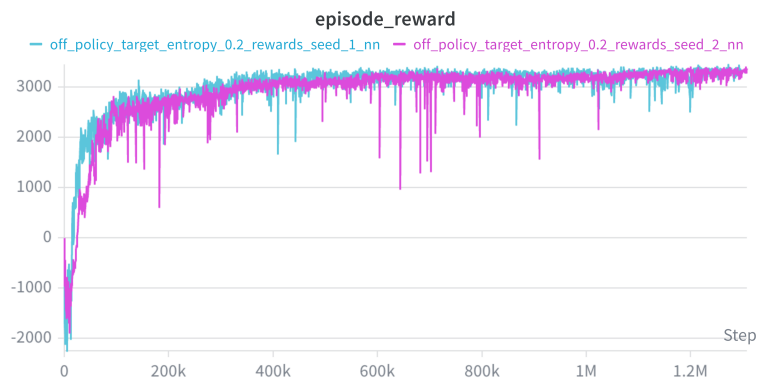


Figure 14. Average Episode Rewards for MASAC w/ with actor network nearest neighbor observation

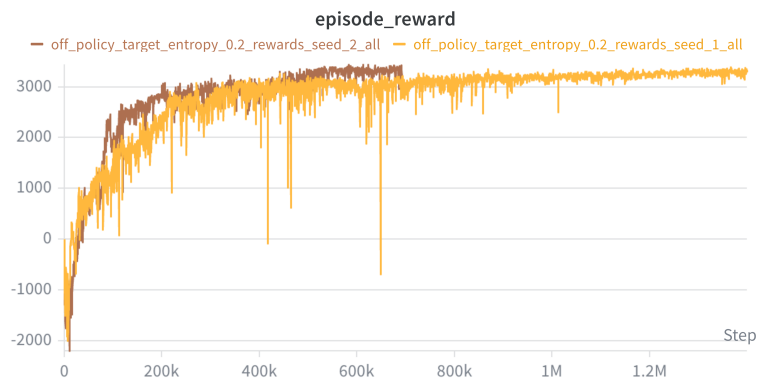


Figure 15. Average Episode Rewards for MASAC w/ with actor network nearest neighbor observation. Note the brown seed was terminated early after learning occurred.

MASAC sensitivity to Reward Scaling

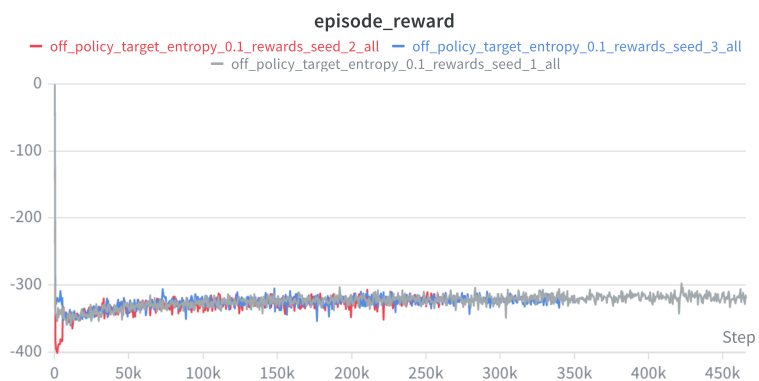


Figure 16. Different reward scaling and reward function tested with MASAC. No learning occurs.

MAPPO Hyperparameters (w/ and w/o Attention)

Table 2. MAPPO Hyperparameters

Parameter	Value
optimizer	Adam
learning rate	$3 \cdot 10^{-4}$
discount (γ)	0.99
GAE (λ)	0.95
rollout buffer size	1024 (global steps)
ϵ -clip	0.2
gradient steps PPO	3

MASAC Hyperparameters

Table 3. MASAC Hyperparameters

Parameter	Value
optimizer	Adam
learning rate	$1 \cdot 10^{-4}$
discount (γ)	0.99
replay buffer size	1000000
α	0.2
critic target τ	0.005

TorchRL Replay Buffer Claude Transcript

Explain all the key components of TorchRL's replay buffer for off-policy algorithms like SAC and Q-learning. Tell me all the components I would need to know about it to use it properly.

TorchRL Replay Buffer
Let me walk through everything you need to know to use TorchRL's replay buffer correctly for off-policy algorithms.

Core Class: `ReplayBuffer`
The main class is `TorchRL.data.ReplayBuffer`. The more common variant for off-policy RL is `TensorDictReplayBuffer`, which expects `TensorDict` objects — this integrates naturally with TorchRL's environment wrappers and collectors.

```
python
from torchrl.data import TensorDictReplayBuffer
from torchrl.data.replay_buffers import LazyMemmapStorage, SamplerWithoutReplacement
```

```
rb = TensorDictReplayBuffer(
    storage=LazyMemmapStorage(max_size=10_000),
    sampler=RandomSampler(),
    batch_size=256,
    transform=None, # optional
)
```

1. Storage
Storage controls **where** and **how** transitions are held in memory.

Class	Description
<code>ListStorage</code>	Python list; flexible but slow and high memory overhead
<code>LazyTensorStorage</code>	RAM-based tensor storage; allocates on first write
<code>LazyMemmapStorage</code>	Memory-mapped storage (backed by disk); ideal for large buffers

`LazyMemmapStorage` is the standard choice for SAC/DQN with large buffers

```
python
from torchrl.data.replay_buffers.storages import LazyMemmapStorage

storage = LazyMemmapStorage(
    max_size=1_000_000,
    scratch_dir="/tmp/rl", # where .mmap files live; unit for RAM memmap
    device=torch.device("cpu"),
)
```

Key behavior: storage is **lazy** — the tensor shapes and dtypes are inferred from the first call to `add()` or `extend()`. This means you must be consistent about the shape of your transitions across all writes.

2. Sampler
The sampler controls **how** indices are drawn during `sample()`.

Class	Description
<code>RandomSampler</code>	Uniform random with replacement (standard SAC/DQN)
<code>SamplerWithoutReplacement</code>	Each transition used once per epoch (more like supervised learning)

Common uses: normalizing observations, converting pixel types, reward scaling. Since transforms run at sample time, the raw data is stored and normalization parameters can be updated dynamically.

5. Adding Data
Single transition: `add()`

```
python
td = TensorDict({
    "observation": obs, # shape [obs_dim]
    "action": action, # shape [act_dim]
    "reward": reward, # shape [1]
    "next_observation": next_obs,
    "done": done,
}, batch_size=1)
idx = rb.add(td)
```

Batch of transitions: `extend()`

```
python
# td has batch_size [1]
rb.extend(td)

extend() is far more efficient when collecting rollouts — use it with a SyncDataCollector that returns batches.
```

6. Sampling

```
python
sample_td = rb.sample() # uses batch_size set at construction
sample_idx = rb.sample(256) # override batch_size

obs = sample_td["observation"]
act = sample_td["action"]
rew = sample_td["reward"]
next_obs = sample_td["next_observation"]
done = sample_td["done"]
```

For PER, the sample also has:

```
python
weights = sample_td["weight"] # IS correction weights, shape [batch_size]
indices = sample_td["index"] # buffer indices, needed for priority update
```

7. Integration with DataCollectors
TorchRL's collectors (e.g., `SyncDataCollector`) produce `TensorDict`s that go directly into the buffer.

```
python
from torchrl.collectors import SyncDataCollector

collector = SyncDataCollector(
    env,
    policy,
    frames_per_batch=1000,
    total_frames=1_000_000,
)
```

`PrioritySampler`: PER — sample proportional to priority P_t

`SliceSampler`: Samples contiguous trajectory slices (for recurrent policies)

RandomSampler (default for SAC)

PrioritySampler (PER)
Implements the standard PER weighting:

$$P(i) = \frac{w_i^\alpha}{\sum_{j \in \mathcal{B}} w_j^\alpha}, w_i = \left(\frac{1}{N} - P(i) \right)^\beta$$

```
python
from torchrl.data.replay_buffers.samplers import PrioritySampler

sampler = PrioritySampler()
```

```
sampler = PrioritySampler(
    min_weight=1e-10, # min. weight
    alpha=0.6, # prioritization exponent
    beta=0.4, # IS correction exponent (often annealed to 1.0)
)
```

After computing new TD errors, you update priorities with:

```
python
rb.update_priority(indices, new_priorities) # indices returned by sample()
```

The sampled `TensorDict` will contain an `"_weight"` key with the IS weights w_i .

3. Writer
The writer governs **how** new transitions are inserted and what gets overwritten when the buffer is full.

Class	Description
<code>RoundRobinWriter</code>	FIFO (circular overwrite) (default)

`TensorDictRoundRobinWriter`: Same, optimized for `TensorDict`

`IncrementalDatasetWriter`: No writes after initial population

For standard SAC/DQN, the default `RoundRobinWriter` (implies) is correct — oldest transitions are overwritten.

4. Transforms
Replay buffer transforms are applied at **sample** time, not at write time — this is important. They are analogous to dataset transforms in PyTorch.

```
python
from torchrl.envs.transforms import ToTensorImage, ObservationNorm
from torchrl.data.replay_buffers import TensorDictReplayBuffer

rb = TensorDictReplayBuffer(
    storage=LazyMemmapStorage(100_000),
    sampler=RandomSampler(),
    transform=Compose(
        ObservationNorm(scale="observation", loc=0.0, scale=1.0),
    ),
)
```

```
For batch, td in collector:
    rb.extend(batch, td)
    if len(rb) >= learning_starts:
        sample = rb.sample()
        loss = loss_module(sample)
        ...
```

8. Key Pitfalls
Shape consistency: Since storage is lazy, the first `add()` / `extend()` call sets the schema. If subsequent transitions have different shapes (e.g., variable-length observations), you'll get an error.

batch_size on the `TensorDict`: The `TensorDict` you add must have a consistent `batch_size`. A single transition should have `batch_size=1` (scalar batch); a rollout buffer should have `batch_size=[1]`.

Device placement: Storage is CPU by default. Sampled batches will be on CPU; move them to GPU explicitly:

```
python
sample = rb.sample().to("cuda")
```

Or set `device="cuda"` on the storage (requires enough VRAM).

PER priority initialization: New transitions need an initial priority. TorchRL defaults to max current priority for new insertions, which is the standard PER heuristic — but verify this matches your implementation intent.

Sampler without replacement: epoch semantics. This sampler exhausts all stored data before repeating. This is unusual for online RL (you'd normally use `RandomSampler`) but useful for offline RL or when treating a collected rollout as a mini-dataset.

Minimal SAC Setup (Putting It Together)

```
python
from torchrl.data import TensorDictReplayBuffer
from torchrl.data.replay_buffers.storages import LazyMemmapStorage
from torchrl.data.replay_buffers.samplers import RandomSampler
```

```
rb = TensorDictReplayBuffer(
    storage=LazyMemmapStorage(max_size=1_000_000),
    sampler=RandomSampler(),
    batch_size=256,
    pin_memory=False,
)

# In training loop:
rb.extend(rollout_td) # add new experience
if len(rb) >= learning_starts:
    batch = rb.sample().to(device) # uniform sample
    loss_td = loss_module(batch)
    optimizer.step()
```

The key insight is that `TensorDictReplayBuffer` is essentially a typed circular buffer store with pluggable sampling strategies — the `TensorDict` format enforces schema consistency and gives you named key access throughout.