On-Policy vs Off-Policy Deep RL in Continuous Control Environments

Varun Rayamajhi
School of Informatics
University of Edinburgh

Kamil Balinski
School of Informatics
University of Edinburgh

I. Introduction and Background

Reinforcement learning (RL) has become a powerful framework for solving sequential decision-making problems. It is widely used in robotics and continuous-control settings. In these scenarios, agents must learn to map high-dimensional observations to continuous actions, such as joint torques or velocities, in order to maximize long-term reward. Deep reinforcement learning methods have shown strong performance on these tasks, especially when combined with simulation environments such as MuJoCo [1].

A key distinction in modern RL algorithms is between on-policy and off-policy learning. These approaches differ in how they collect and reuse training data, leading to important performance trade-offs. In this work, we compare Proximal Policy Optimisation (PPO) [2], a widely used on-policy algorithm, with Soft Actor-Critic (SAC) [3], a state-of-the-art off-policy method for continuous control.

A. On-Policy vs Off-Policy RL

On-policy methods require that the current policy generate training data. This typically leads to more stable updates, since the training distribution matches the policy being optimized. However, these methods are less sample efficient, as previously collected data cannot be reused extensively.

Off-policy methods, in contrast, store transitions in a replay buffer and reuse them during training. This improves sample efficiency, but can introduce instability due to the mismatch between past and current policies.

This trade-off between stability and sample efficiency is central to modern reinforcement learning and motivates the comparison between PPO and SAC in this work.

B. Proximal Policy Optimization

Proximal Policy Optimisation (PPO) is an on-policy actor-critic algorithm designed to improve the stability of policy-gradient methods. PPO constrains policy updates using a clipped objective based on the importance-sampling ratio

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

Intuitively, this clipping prevents the policy from changing too much in a single update, which reduces the risk of unstable or destructive updates. This helps explain why PPO tends to produce more stable learning behaviour in practice.

In our implementation, PPO uses separate actor and critic networks, each with two hidden layers of 64 units. The policy is modelled as a Gaussian distribution over continuous actions, and advantages are estimated using Generalized Advantage Estimation (GAE). PPO is widely used due to its simplicity and stable optimisation behaviour.

C. Soft Actor-Critic

Soft Actor-Critic (SAC) is an off-policy actor-critic algorithm that incorporates entropy regularization into the objective. SAC maximizes both expected return and policy entropy, encouraging exploration.

The critic target is given by

$$y_t = r_t + \gamma(1 - d_t)\left(\min_i Q(s', a') - \alpha \log \pi(a'|s')\right),$$

and the actor is trained to maximize expected Q-values while maintaining entropy.

Intuitively, the entropy term encourages the agent to remain uncertain and explore a wider range of actions, rather than converging too quickly to a narrow policy. This is particularly useful in continuous-control tasks where good behaviours may be difficult to discover.

Our implementation uses two Q-networks, target networks updated via Polyak averaging, and a replay buffer. The actor and critic networks use two hidden layers of 256 units. Automatic entropy tuning adapts exploration during training.

D. Relevance to Robotics

Continuous-control RL is closely tied to robotics, where actions correspond to physical control signals such as torques or velocities. PPO is often used because it is stable, while SAC is appealing for its sample efficiency. This trade-off matters a lot in robotics, where gathering real-world data can be costly and time-consuming. Because of this, both PPO and SAC have become common baseline methods in robotic reinforcement learning. This trend is evident in recent applications like autonomous driving, where effective learning and dependable control are essential. [4]

II. Experimental Setup

A. Environments and MDP Formulation

We evaluate PPO and SAC on two MuJoCo environments: HalfCheetah-v4 and Pusher-v4 [1] [5]. HalfCheetah is a planar locomotion task where the agent must learn to run forward [6],

while Pusher is a manipulation task where a robotic arm must move an object towards a target location [7]. We selected these environments to represent two different types of robotic tasks: locomotion and manipulation. More broadly, they capture two different challenges in robotics. HalfCheetah focuses on coordinated locomotion with relatively smooth dynamics, while Pusher involves more precise manipulation and interaction with objects. This allows us to evaluate how PPO and SAC perform across different types of control problems.

Both environments can be formulated as Markov Decision Processes (MDPs), defined by states, actions, transition dynamics, rewards, and a discount factor. At each timestep, the agent observes the current state, selects an action, and receives a reward signal.

In HalfCheetah, the observation space consists of joint positions and velocities, and the action space corresponds to continuous joint torques. The reward encourages forward velocity while penalising excessive control. In Pusher, the observation includes the robot state, object position, and goal location. The action space consists of continuous torques applied to the joints, and the reward encourages the object to move closer to the target.

These environments were chosen for their relevance to real-world robotics. HalfCheetah resembles locomotion in quadruped robots, while Pusher reflects industrial manipulation tasks requiring precise object control. This makes them suitable for comparing PPO and SAC, as PPO benefits from stable dynamics, whereas SAC may perform better on tasks requiring exploration and fine control.

B. Training and Evaluation Protocol

We used a two-phase evaluation procedure. First, we performed single-seed sweeps to select the best hyperparameter setting for each algorithm and environment. We then evaluated the selected settings across five seeds. PPO was trained for $1M$ steps, but SAC was trained for $400k$ steps to keep the runtime manageable. In the final comparison, we used mean episodic return together with learning curves and seed-wise performance distributions.

III. Hyperparameter Tuning

A. Choice of Hyperparameters

1) PPO: For PPO, we adjusted the learning rate and clipping coefficient because these had the biggest impact on performance. We looked at other hyperparameters, but none led to consistent improvements.

The learning rate affects how quickly the policy updates, which in turn affects both how fast it converges and its stability. The clipping coefficient controls the size of policy updates and helps prevent large, unstable changes. In practice, we found that the learning rate had a bigger impact on performance. Clipping mainly affected the smoothness of the training.

2) SAC: For SAC, we tuned the critic learning rate and the target network update rate (τ), as both had a strong impact on training behaviour.

The critic learning rate affects how quickly Q-values are updated, which is important for guiding policy learning. If set too low, the agent struggles to leverage past experience, while values that are too high can lead to unstable estimates. The parameter τ controls the update rate of the target networks via Polyak averaging. Smaller values improve stability but slow down adaptation. Larger values allow faster learning, but they also increase instability. In practice, both parameters were important for balancing stability and learning efficiency.

B. Grid Search Results

Figure 1 shows the results of our hyperparameter sweeps for PPO and SAC on both environments. Higher values indicate better performance.[1] We use these results to identify the best-performing setting for each algorithm and environment, and to examine how sensitive each method is to its hyperparameters.

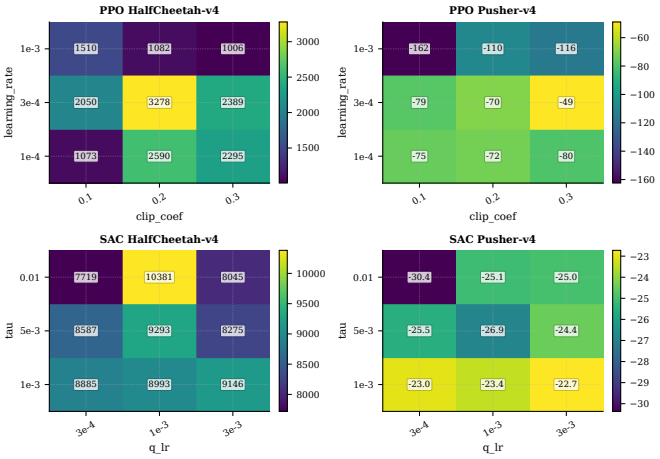

Fig. 1. Grid-search results for PPO and SAC on both environments. Each cell reports mean return over the last 50 episodes for one hyperparameter setting.

1) PPO Sweep Results: For PPO, the learning rate had the clearest effect in both environments. For a fixed clip coefficient, performance improved when the learning rate increased from 1×10^{-4} to 3×10^{-4}, but then dropped again at 1×10^{-3}. This suggests that 1×10^{-4} was too conservative within the training budget, while 1×10^{-3} produced updates that were too aggressive. In both HalfCheetah-v4 and Pusher-v4, the best learning rate was 3×10^{-4}.

The effect of the clip coefficient was weaker and depended more on the environment. In HalfCheetah-v4, the best result was obtained with clip coefficient 0.2, while in Pusher-v4 the best result was obtained with clip coefficient 0.3. This suggests that PPO was more sensitive to the learning rate than to the clip coefficient (however, the clipping choice still affected final performance). Overall, PPO appeared relatively

[1] In Pusher-v4, returns are negative, so values closer to zero are better.

robust to hyperparameter choice, since the same learning rate worked best across both environments. Based on these results, we selected lr $= 3 \times 10^{-4}$, clip $= 0.2$ for HalfCheetah-v4 and lr $= 3 \times 10^{-4}$, clip $= 0.3$ for Pusher-v4.

2) SAC Sweep Results: For SAC, performance depended on both the critic learning rate and the target-network update rate τ. In both environments, the smallest critic learning rate, 3×10^{-4}, appeared to give weaker performance, especially when combined with a larger τ. This suggests that if the critic learns too slowly, SAC cannot make full use of replayed experience.

The best SAC setting was different across the two environments. In HalfCheetah-v4, the best result was obtained with $\tau = 0.01$ and $q\_lr = 10^{-3}$. In Pusher-v4, the best result was obtained with $\tau = 10^{-3}$ and $q\_lr = 3 \times 10^{-3}$. This indicates that SAC was more environment-dependent than PPO in our sweeps. It also indicates that SAC was more sensitive to hyperparameter choice, since strong performance appeared only for a smaller number of settings and didn't follow a clear pattern as PPO. Based on these results, we selected $\tau = 0.01$, $q\_lr = 10^{-3}$ for HalfCheetah-v4 and $\tau = 10^{-3}$, $q\_lr = 3 \times 10^{-3}$ for Pusher-v4.

IV. Results

A. Quantitative Comparison

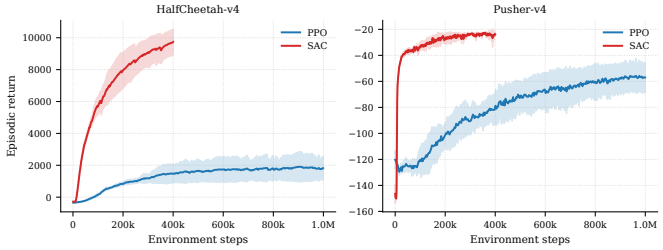

Fig. 2. Mean episodic return versus environment steps for PPO and SAC on HalfCheetah-v4 and Pusher-v4 across 5 seeds. Shaded regions indicate ± 1 standard deviation.

Across both environments, PPO and SAC showed clear differences in sample efficiency, training stability, wall-clock training time, and reliability across seeds.

In terms of sample efficiency, SAC outperformed PPO in both environments. In HalfCheetah-v4, SAC reached roughly $10k$ episodic returns by around $400k$ steps, whereas PPO was still near $2k$ returns even after $1M$ steps. In Pusher-v4, SAC improved to around -40 within the first $50k$ steps and later converged near -20 to -25. On the other hand, PPO improved much more slowly and only reached around -60 by the end of training. This suggests that SAC made better use of each interaction with the environment, which is consistent with its off-policy design and replay buffer.

In terms of training stability, SAC generally showeed smoother learning curves than PPO. However, this depended somewhat on the environment. In Pusher-v4, after its initial jump from around -150 to around -40, SAC remained relatively steady and gradually improved toward -20. On the other hand, PPO improved from roughly -120 to -60 more slowly and with more visible variation throughout training. In HalfCheetah-v4, SAC's mean return increased fairly smoothly from near 0 to near $10k$. However, PPO showed slower progress and more variability, with returns spread much more broadly later in the training.

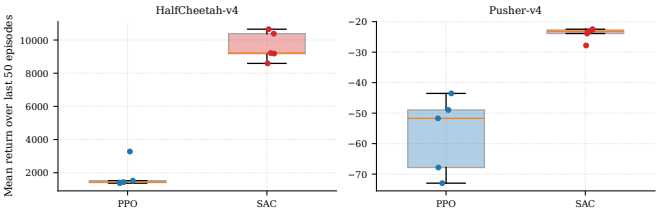

Fig. 3. Distribution of final performance across 5 seeds for PPO and SAC on HalfCheetah-v4 and Pusher-v4. Each boxplot summarizes the mean return over the last 50 episodes of a final run. Points denote individual seeds.

Across seeds, SAC also showed more consistent final performance overall (Fig. 3). In both environments, SAC achieved higher final performance across seeds, and in Pusher-v4 it also showed a tighter spread than PPO. This suggests that SAC gave more consistent final results.

In terms of wall-clock time, PPO was cheaper to run in practice. Our hyperparameter sweeps took about 5 hours for PPO and about 9.5 hours for SAC on our setup. Therefore, although SAC was more sample-efficient, it also required more computation time. We note that these two methods were not run under a perfectly matched training budget, so we treat this as a practical runtime comparison rather than a fully controlled benchmark.

B. Qualitative Policy Behavior

To support the quantitative results, we looked at the behavior of the trained policies through rendered rollouts. Overall, SAC produced smoother, more consistent behaviour across both environments, whereas PPO showed greater variability between runs.

In HalfCheetah-v4, SAC consistently learned a smooth and stable running gait across all seeds, maintaining forward motion with little variation. In contrast, PPO behaviour varied significantly. In one seed, the agent moved forward as intended, though the motion was less smooth than SAC's. However, in the remaining runs, the agent often failed to learn a stable gait, frequently flipping upside down and continuing to move in an unstable or unintended manner.

In Pusher-v4, SAC again showed consistent behaviour across seeds, reliably pushing the object towards the target with controlled and purposeful movements. PPO, however, was less reliable. A small number of seeds (1 and 5) moved the object close to the target, but most runs failed to complete the task, often interacting with the object in an unintended way or moving it indirectly through unstable arm motion rather than controlled pushing.

This variability may partly be due to PPO being tuned with a single seed. As a result, the selected hyperparameters may be well-suited to that specific training trajectory, but less robust

across different initialisations. In contrast, SAC appears to produce more consistent policies across seeds, likely due to its off-policy nature and use of replayed experience.

V. Proposed Robotics Task

We consider a **multi-agent formation control task with safety constraints** in a dynamic environment. A team of mobile robots must maintain a desired formation while tracking a moving goal and avoiding collisions with teammates and dynamic obstacles. We find this problem interesting because it combines continuous control, coordination, safety, and uncertainty. It is also closer to real robotic systems or tasks such as search-and-rescue than standard single-agent benchmark tasks.

At first, this problem may seem solvable using only classical control, since formation control, tracking, and collision avoidance are all well-studied problems [8]. However, the task becomes difficult in dynamic environments, where robots must adapt their behavior online. For example, if the robots need to pass through a narrow passage, they may have to temporarily compress or deform the formation, move through the passage safely, and then recover the original formation afterward. Because these behaviors depend on the environment, they are not easy to design by hand. This makes RL useful for this task. However, since this is a safety-critical multi-agent task, RL alone is not sufficient. We would still need reliable low-level control and safety constraints.

A. Environment Formulation

Because each robot only has access to local and noisy sensor information rather than the full system state, we would model this problem as a partially observable Markov decision process (POMDP): $(\mathcal{S}, \mathcal{O}, \mathcal{A}, P, \Omega, R, \gamma)$ [9].

The full state $s \in \mathcal{S}$ contains the poses of all robots, the states of dynamic obstacles, and the goal state. Each robot follows unicycle dynamics: $\dot{x}_i = v_i \cos\theta_i, \quad \dot{y}_i = v_i \sin\theta_i, \quad \dot{\theta}_i = \omega_i$, where (x_i, y_i, θ_i) are the pose variables of robot i, and (v_i, ω_i) are its linear and angular velocity commands.

Since each robot only has local sensing, the observation $o_i \in \mathcal{O}$ would include the relative positions of nearby robots, distances to nearby obstacles, and the local direction of the goal. The action for each robot is continuous $a_i = (v_i, \omega_i)$. The transition model P is determined by the robot dynamics and the environment, and the observation model Ω maps the true state to noisy local observations.

Given the requirements of our proposed task, we define the following reward to balance formation accuracy, goal progress, safety, and control effort:

$$r = -\lambda_f \sum_{(i,j)\in\mathcal{E}} (d_{ij} - d_{ij}^{\text{target}})^2 - \lambda_g \|x_{\text{leader}} - x_{\text{goal}}\|^2 - \lambda_s \sum_i \phi_i^{\text{safe}} - \lambda_c \sum_i \mathbb{I}(\text{collision}_i) - \lambda_u \sum_i \|a_i\|^2$$

The first term encourages formation maintenance, the second rewards progress toward the goal, the third penalizes unsafe proximity to obstacles or teammates, the fourth penalizes collisions, and the last penalizes overly aggressive control. An episode ends when the team reaches the goal successfully or when a collision occurs.

B. Algorithm Selection and Justification

Between PPO and SAC, we would choose SAC for this task. Our proposed task is a continuous-control problem in a complex environment, so sample efficiency matters. Since SAC is off-policy, it can reuse past experience through the replay buffer, which makes it more data-efficient than PPO. This is especially useful in multi-robot tasks, where learning good coordination may require many interactions.

Another advantage of SAC is that its entropy regularization encourages exploration. This is useful in our task because the robots may need to learn different ways to adjust the formation when they face obstacles or noisy sensing. We could also use PPO, but since it's on-policy, it discards older samples after each update. This makes it less appealing for a more difficult multi-agent setting.

However, we would not use SAC in its standard form. Instead, we would use a centralized-training, decentralized-execution setup [10]. During training, the method can use more global information to make learning more stable, but during execution, each robot acts only from its own local observations. This is more realistic for real deployment. More importantly, we would not rely on RL alone for safety. Instead, we would use RL for high-level adaptation and coordination, and use structured control methods for low-level safety and tracking.

C. Challenges and Alternatives

This task introduces several important challenges. Safety is of the main challenges, since a learned policy may still occasionally produce unsafe actions. Partial observability also makes the task difficult because each robot only has access to part of the environment. In addition, multi-agent non-stationarity appears because all agents learn at the same time, so the environment seen by each agent keeps changing during training. This can make learning unstable and make past experience less useful [11]. Finally, there is a clear sim-to-real gap, since a policy trained in simulation may behave differently on real robots because of sensing noise, delays, or modeling errors.

For these reasons, we would not choose pure RL as the final solution. Instead, we propose a more realistic hybrid architecture: RL handles higher-level coordination decisions, such as when to relax the formation or how aggressively to move toward the goal, and a lower-level controller handles safety and tracking. To enforce safety in this architecture, we could combine the learned policy with Control Barrier Functions (CBFs) [12], which can filter the nominal action to keep the system inside a safe set: $\dot{h}(x,u) + \alpha h(x) \geq 0$. Another option is to use Model Predictive Control (MPC) [13] for reliable short-horizon planning and control. Imitation learning could also help by initializing the policy from expert

demonstrations, although its performance would still depend on the quality of those demonstrations.

Overall, we think this task is a good example of a setting where RL is useful but not sufficient on its own. RL can help learn adaptive multi-agent coordination strategies that are difficult to design by hand, but classical control and safety filters are still needed to make the system reliable for real-world deployment.

REFERENCES

[1] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 5026–5033.

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

[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.

[4] A. J. M. Muzahid, S. F. Kamarulzaman, and M. A. Rahman, "Comparison of ppo and sac algorithms towards decision making strategies for collision avoidance among multiple autonomous vehicles," in *2021 International Conference on Software Engineering & Computer Systems and 4th International Conference on Computational Science and Information Management (ICSECS-ICOCSIM)*, 2021, pp. 200–205.

[5] M. Towers, A. Kwiatkowski, J. Terry *et al.*, "Gymnasium: A standard interface for reinforcement learning environments," *arXiv preprint arXiv:2407.17032*, 2024.

[6] Farama Foundation, "Half cheetah - gymnasium documentation," https://gymnasium.farama.org/v0.29.0/environments/mujoco/half_cheetah/, accessed: 2026-04-02.

[7] ——, "Pusher - gymnasium documentation," https://gymnasium.farama.org/v0.29.0/environments/mujoco/pusher/, accessed: 2026-04-02.

[8] M. Mesbahi and M. Egerstedt, "Graph theoretic methods in multiagent networks," 2010.

[9] M. Lauri, D. Hsu, and J. Pajarinen, "Partially observable markov decision processes in robotics: A survey," *IEEE Transactions on Robotics*, vol. 39, no. 1, pp. 21–40, 2022.

[10] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," *Advances in neural information processing systems*, vol. 30, 2017.

[11] G. Papoudakis, F. Christianos, A. Rahman, and S. V. Albrecht, "Dealing with non-stationarity in multi-agent deep reinforcement learning," *arXiv preprint arXiv:1906.04737*, 2019.

[12] A. D. Ames, X. Xu, J. W. Grizzle, and P. Tabuada, "Control barrier function based quadratic programs for safety critical systems," *IEEE Transactions on Automatic Control*, vol. 62, no. 8, pp. 3861–3876, 2016.

[13] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, no. 6, pp. 789–814, 2000.