(Evolutionary) Proximal Policy Optimization for Dexterous Robotic Reach Tasks

Emil Kasper

January 28, 2026

Abstract

This project investigates Proximal Policy Optimization (PPO) and an evolutionary extension, Evolutionary PPO (EPPO), on a dexterous robotic reaching task based on the FetchReachDense environment. The goal is to study whether evolutionary population-based updates can match or complement standard policy-gradient learning in continuous control settings. Implementation choices and comparisons against a random baseline are presented, along with results that show fast and smooth convergence for PPO, and comarable results for EPPO.

Disclaimer

This report is not a scientific paper. It presents experimental results obtained from two custom implementations of PPO and a modified evolutionary variant, evaluated in a continuous robotics environment. The work is exploratory in nature and does not aim to provide novel theoretical contributions or exhaustive empirical validation.

1 Introduction

Reinforcement learning has proven effective for continuous control problems, particularly when combined with stable policy-gradient methods such as Proximal Policy Optimization (PPO). Genetic methods such as Neuroevolution of Augmenting topologies (NEAT) have been proposed in the past as an alternative to reinforcement leanring algorithms.

In this project, a hybrid Evolutionary PPO (EPPO) approach that combines PPO updates with evolutionary selection, along with a classic PPO algorithm are examined. It is further shown with the weak performance of a random baseline that success is not trivial.

2 Environment Description

All algorithms are evaluated on the FetchReach environment from Gymnasium Robotics (link). The task models a simulated robotic manipulation scenario in which a 7-DoF robotic arm is required to move its end-effector to a randomly sampled target position in three-dimensional space.

2.1 Task Overview

The objective of the FetchReach task is to position the robot's end-effector (gripper) as close as possible to a desired goal location. A successful episode is defined by the Euclidean distance between the achieved end-effector position and the target position falling below a predefined success threshold. In this project, the proposed threshold of 0.05, as well as a more tighter threshold of 0.02 are examined.

2.2 Action Space

The action space is continuous and four-dimensional:

$$\mathcal{A} = [-1, 1]^4$$

The first three components correspond to Cartesian displacements of the end-effector along the x, y, and z axes. The fourth component controls the gripper opening. However, in the FetchReach task this component is not used and remains effectively constant.

2.3 Observation and State Representation

At each time step, the environment provides observations containing three components:

- `observation`
- `achieved_goal`
- `desired_goal`

Observation Vector: The `observation` component contains the physical state of the robot relevant for control. This includes:

- End-effector position (x, y, z)
- End-effector linear velocity
- Gripper joint positions
- Gripper joint velocities

Goal Information The `achieved_goal` corresponds to the current Cartesian position of the end-effector, while `desired_goal` represents the target position sampled at the beginning of each episode. Both are three-dimensional vectors.

In this implementation, all components are concatenated into a single flat observation vector before being passed to the policy network.

2.4 Initial State Distribution

At the beginning of each episode, the robotic arm is initialized in a fixed neutral configuration. The desired goal position is sampled uniformly within a predefined workspace volume around the robot. As a result, while the starting robot configuration remains constant, the task difficulty varies due to different target positions.

2.5 Simulation and Control Frequency

The environment is simulated using `MuJoCo` with a simulation time step of $dt = 0.002\,\text{s}$. Control actions are applied at a lower control frequency, with each agent action being held constant for multiple simulation steps.

2.6 Episode Termination and Reward

An episode terminates either when the success condition is met or when a fixed maximum number of time steps is reached. In FetchReachDense configuration used for this project the reward is based on the negative euclidean distance between the achieved and desired goal positions and encourages minimizing this distance. An additional success bonus is provided when the target is reached.

3 Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) optimizes a stochastic policy by maximizing a surrogate objective that balances policy improvement, value function accuracy, and exploration. The overall objective consists of the following three components:

3.1 Clipped Policy Objective

The core of PPO is the clipped surrogate policy objective

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

where

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

is the probability ratio between the current and the previous policy, and $\hat{A}_t$ denotes the advantage estimate computed using Generalized Advantage Estimation (GAE). The clipping operation prevents excessively large policy updates and stabilizes training.

3.2 Value Function Loss

To ensure accurate value estimates, PPO includes a value function loss term. In the implementation, a clipped value loss proposed in the PPO paper is used:

$$L^{\text{VF}}(\theta) = \mathbb{E}_t \left[\max \left(\left(V_\theta(s_t) - \hat{R}_t \right)^2,\ \left(V_{\text{clip}}(s_t) - \hat{R}_t \right)^2 \right) \right], \tag{2}$$

where

$$V_{\text{clip}}(s_t) = V_{\theta_{\text{old}}}(s_t) + \text{clip}(V_\theta(s_t) - V_{\theta_{\text{old}}}(s_t), -\epsilon_v, \epsilon_v)\,,$$

and $\hat{R}_t$ denotes the empirical return. This clipping mechanism prevents large and destabilizing updates of the critic.

3.3 Entropy Regularization

To encourage exploration an entropy bonus is added:

$$\mathcal{H}(\pi_\theta(\cdot \mid s_t)) = -\mathbb{E}_{a\sim\pi_\theta} \left[\log \pi_\theta(a \mid s_t) \right]. \tag{3}$$

3.4 Final PPO Objective

Combining all three components, the PPO objective is defined as

$$J^{\text{PPO}}(\theta) = \mathbb{E}_t \left[L^{\text{CLIP}}(\theta) - c_1 L^{\text{VF}}(\theta) + c_2 \mathcal{H}(\pi_\theta(\cdot \mid s_t)) \right], \tag{4}$$

where c_1 and c_2 are scalar coefficients weighting the value loss and entropy bonus.

In practice, optimization is performed by minimizing the corresponding loss function

$$\mathcal{L}^{\text{PPO}}(\theta) = -L^{\text{CLIP}}(\theta) + c_1 L^{\text{VF}}(\theta) - c_2 \mathcal{H}(\pi_\theta(\cdot \mid s_t)), \tag{5}$$

which is equivalent to maximizing $J^{\text{PPO}}(\theta)$.

3.5 Implementation Details

PPO is implemnted following the clipped surrogate objective introduced in the original paper [3]. As algorithmic and implementation details and may vary, I will briefly describe important design choices. They are justified by either following the original PPO implementation or the attempt to improve stability and sample efficiency in continuous-control tasks.

Clipping Additionally to the known policy clipping, the approximate KL divergence between the old and new policies is monitored during training. When the average KL exceeds a threshold, further optimization on the current batch is stopped.

Shared Actor-Critic Architecture: The policy (actor) and value function (critic) share a common feature-extraction network, with separate output heads for action selection and value prediction. This is motivated by the observation that both components rely on similar high-level representations of the environment state (e.g. velocity). Sharing parameters should allow the actor and critic to benefit from each other's learned representations.

Policy Parameterization: For continuous bounded action spaces, the policy is parameterized as a Gaussian distribution followed by a tanh squashing function. This ensures that sampled actions respect the environment's action bounds of [-1,1]. This idea was introduced by [2] in the context of Soft Actor-Critic.

Advantage Estimation: Furhtermore, I use Generalized Advantage Estimation (GAE) to reduce the variance of policy gradient estimates by combining multi-step TD errors, while introducing a controlled amount of bias. For that, I use a discount factor of $\gamma = 0.99$ and a GAE parameter of $\lambda = 0.95$.

Value Function Learning: The value function is trained using a clipped value loss instead of a simple value loss mentioned in the original PPO paper. This, however, is consistent with the implementation details accompanying the original PPO algorithm. Value clipping constrains the magnitude of critic updates between successive policy iterations. I figured this would be especially beneficial as actor and critic share parameters. Although later work has shown that value clipping can be disadvantageous in some circumstances [1], I wanted to remain faithful to the original PPO implementation as much as possible.

Training Setup: Rollouts consist of a fixed batch of 4096 environment steps collected per epoch. Training is performed for 400 epochs, corresponding to approximately 1.5×10^6 environment interactions. No separate evaluation environment is used, therefore all reported metrics correspond to training performance.

4 Evolutionary PPO (EPPO)

4.1 Motivation

EPPO is designed to explore whether population-based selection can guide learning when combined with PPO updates. Instead of a single policy, EPPO maintains a population of policies that are periodically selected and mutated.

4.2 Algorithm Overview

Each generation proceeds as follows:

1. Maintain a population of $N = 8$ policies
2. Train each policy using PPO for a fixed number of steps
3. Evaluate all policies
4. Select the top $K = 4$ elites using lexicographic fitness:

    - Success rate
    - Minimum distance to goal
    - Episode return

5. Generate offspring by mutating elite policies

This process is repeated for multiple generations until the total number of environment steps matches PPO training.

Mutation Offspring are generated by copying the parameters of randomly selected elite individuals and applying Gaussian noise. Network weights are perturbed with standard deviation σ_w, while the exploration parameter $\log \sigma$ is mutated separately using $\sigma_{\log \sigma}$. After mutation, $\log \sigma$ is clamped to a predefined range to ensure stable exploration. Elite individuals are copied unchanged into the next generation.

Training vs. Evaluation Separation EPPO explicitly separates training and evaluation. Training metrics are computed from rollouts used to update PPO, while evaluation rollouts are used solely to assess fitness. Evaluation data is never reused for training, ensuring unbiased fitness estimation.

4.3 Design Choices

Key design decisions include:

- Mutation applied only to offspring (not elites)
- Fixed PPO hyperparameters across the population
- Separate training and evaluation environments for EPPO

Unlike standard evolutionary strategies, EPPO still relies on PPO updates for within-generation learning.

5 Baselines

A random policy baseline is included for reference. Actions are sampled directly from the environment action space without learning. This baseline highlights the difficulty of the task and provides a lower bound on performance.

6 Results

6.1 PPO Performance

PPO converges after roughly 100,000 - 200,000 environment steps, reaching near-perfect success rates and steadily reducing the distance to the goal, indicating successful optimization. Success convergence for a threshold of 0.05 requires slightly less than 100,000 steps, while success convergence for a threshold of 0.02 requires 150,000 steps. Plots can be seen in Fig 1 and Fig 2.

6.2 EPPO Performance

EPPO achieves good performance for a threshold of 0.05, with success rates converging to 1.0 and goal distances comparable to PPO. However, for the run with threshold 0.02, success and total loss do not show sure converge to the optimum. In general, learning progresses more slowly requiring about 500,000-1,000,000 environment steps. Plots can be seen in Fig 3 and Fig 4.

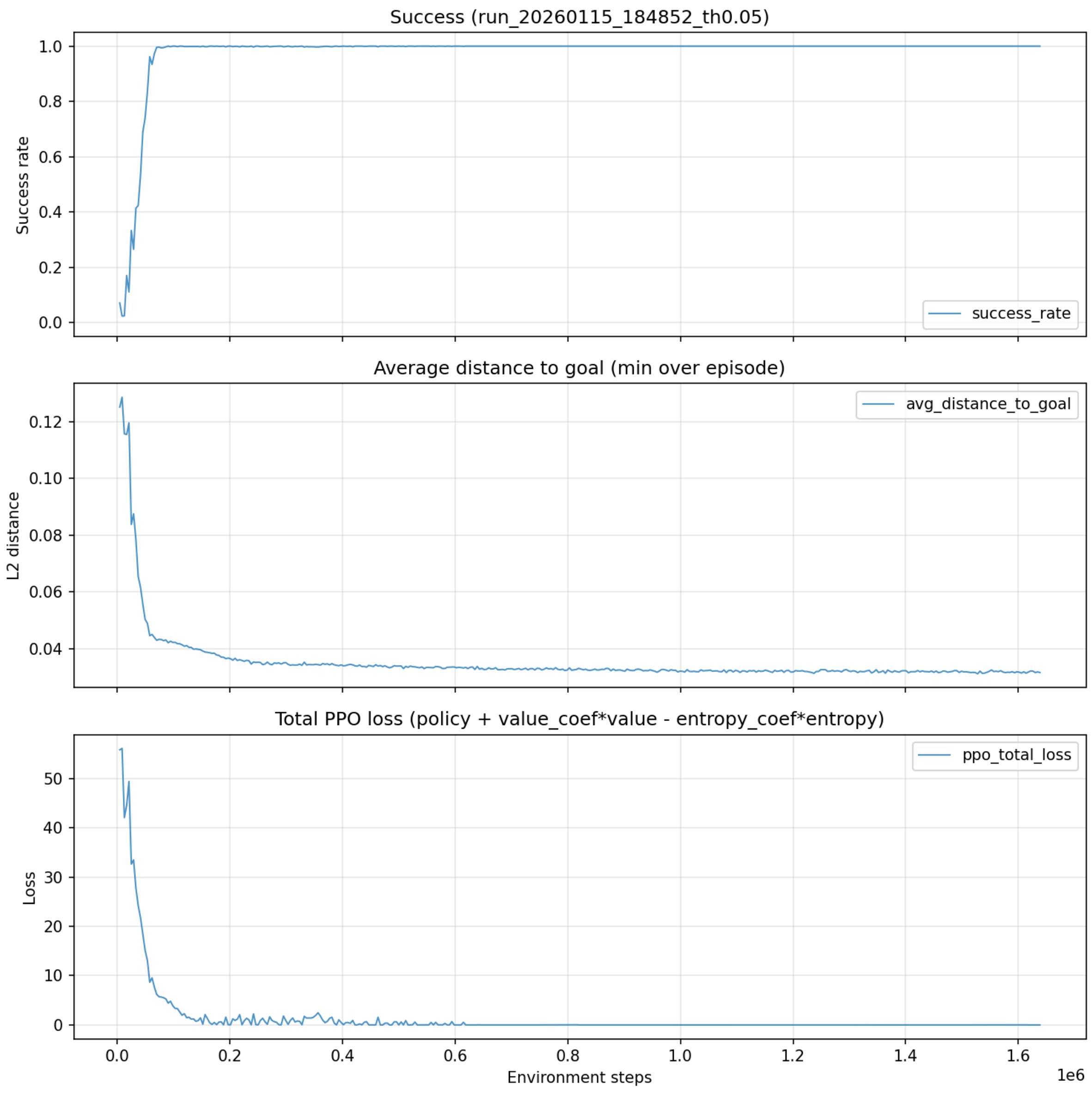

Figure 1: PPO training curves with a threshold of 0.05

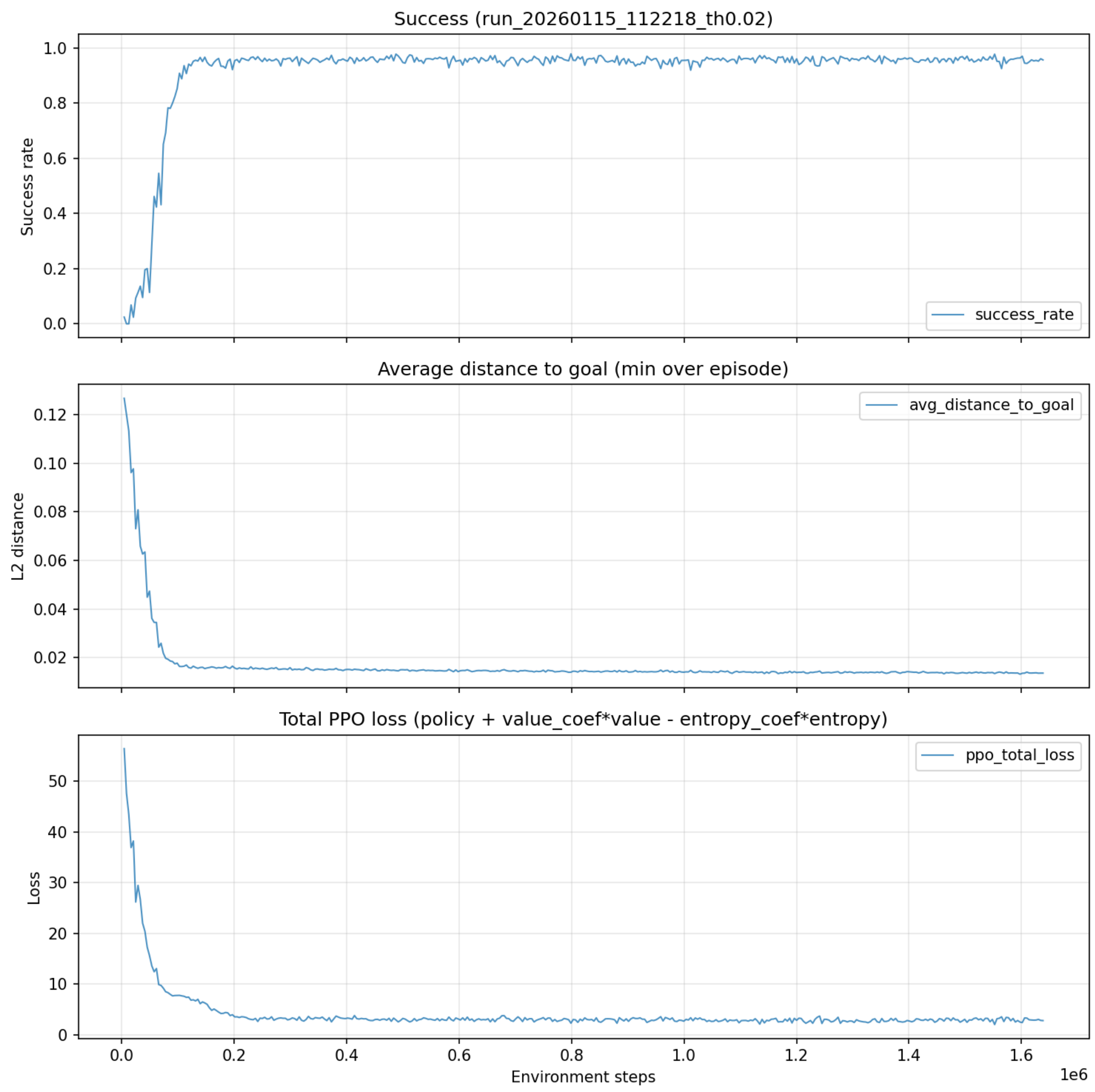

Figure 2: PPO training curves with a threshold of 0.02

6.3 Random Baseline

The random baseline consistently fails to solve the task, maintaining low success rates and large goal distances. This confirms that the task is non-trivial and requires actual learning algorithms. Plots can be seen in Fig 5 and Fig 6.

7 Discussion

The results demonstrate that PPO remains the most efficient and stable method for this task. Compared to PPO, EPPO achieves comparable final performance despite requiring more environment steps, offering a different learning paradigm that may perhaps be useful in certain settings.

Importantly, the random baseline highlights that both PPO and EPPO learn meaningful control policies rather than exploiting trivial dynamics.

Experiments with a stricter success threshold (0.02) yield qualitatively similar results for both PPO and EPPO, indicating robustness across task difficulty.

It has to be noted that these results are not scientifically sound, but serve as a short comparison of two algorithms on a fairly simple environment. Of course, implementation details and hyperparameter selection can have a major impact on learning outcomes,

References

[1] Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphaël Marinier, Leonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. What matters for on-policy deep actor-critic methods? a large-scale study. In *International Conference on Learning Representations*, 2021.

[2] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

[3] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

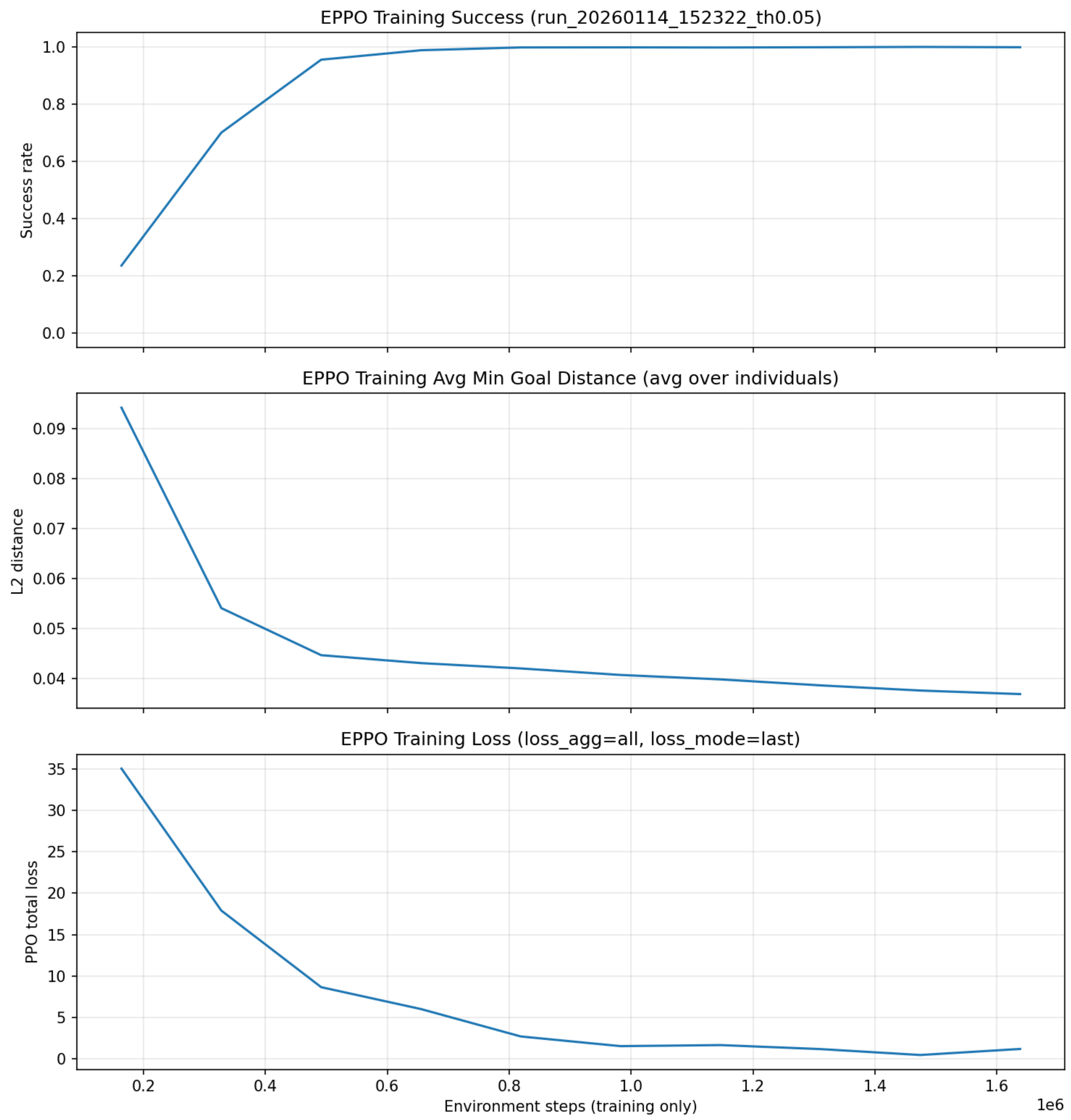

Figure 3: EPPO training curves with a threshold of 0.05

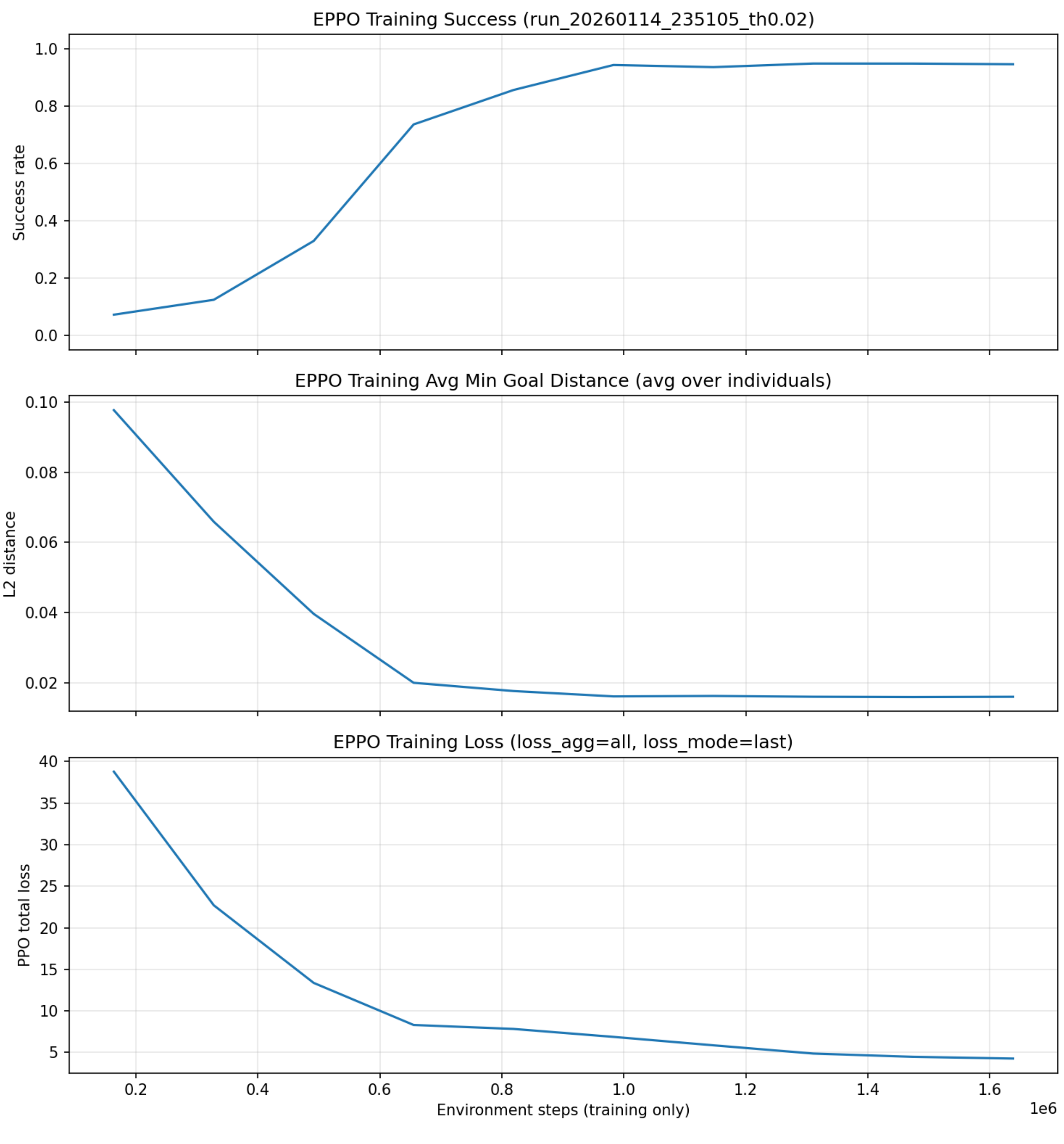

Figure 4: EPPO training curves with a threshold of 0.02

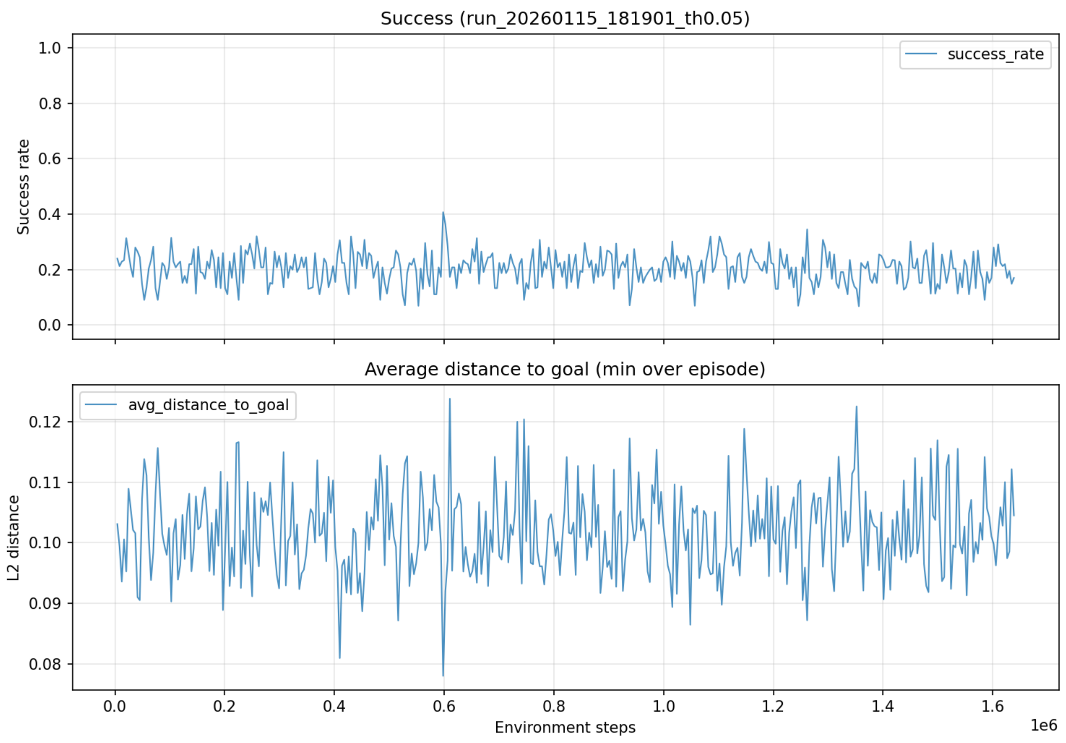

Figure 5: Random baseline with a threshold of 0.05

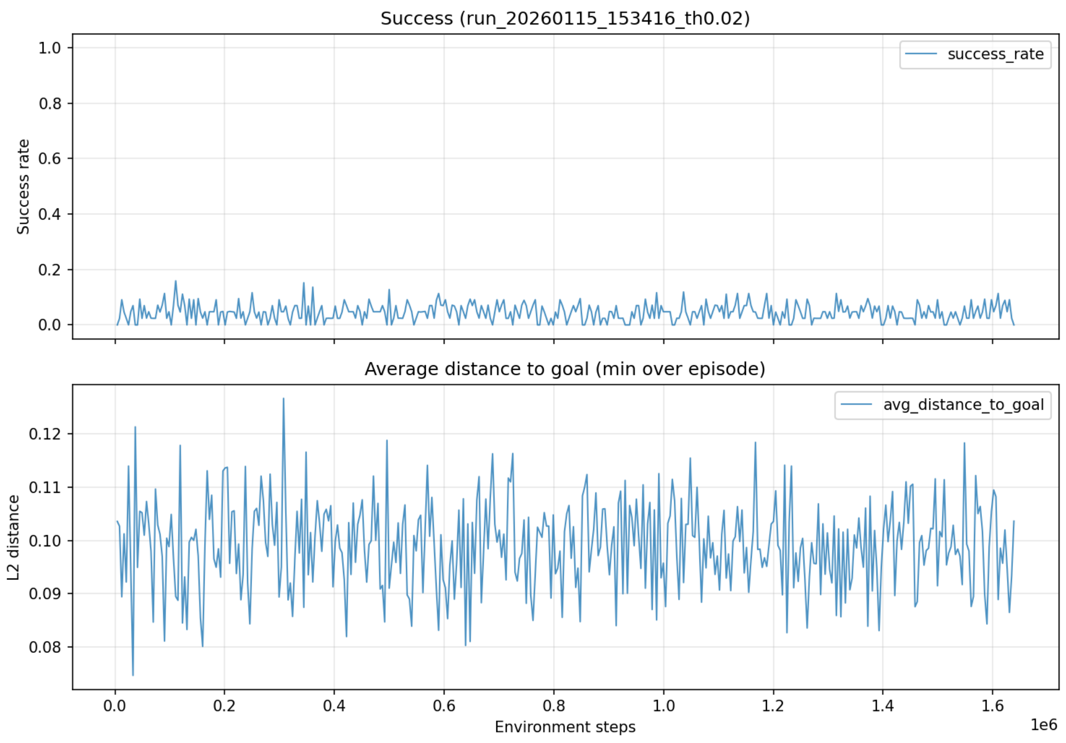

Figure 6: Random baseline with a threshold of 0.02