Reducing Reward Variance of Continuous-Control DRL Agents via Genetic Algorithm Pre-Training

Caleb Dame, James Williams
Johns Hopkins University – Deep Learning Developments with PyTorch

Introduction

Canonical deep reinforcement learning (DRL) methods such as Proximal Policy Optimization (PPO) (Schulman et al. [2017]) and newer methods like Soft Actor-Critic (SAC) (Haarnoja et al. [2018]) can achieve strong results in continuous control domains; however, they are often brittle in that training can be unstable and highly sensitive to hyperparameters and the random seed for weight initialization. In contrast, genetic algorithms (GAs) provide a non-gradient-based population search technique that is more robust in early exploration but computationally inefficient in fine-tuning. This project investigates whether using a GA to pre-train policy weights can reduce reward variance and improve DRL stability and convergence when training from scratch.

Our experimental design compares baseline DRL agents initialized with random weights against agents initialized using GA-evolved policies across several MuJoCo continuous-control environments. The immediate findings of this effort suggest that GA-initialized policies may actually tend to under-perform baselines in reward metrics and contribute in no consistent way to lowering the "failure rate" of a DRL agent.

Environments

Our experiments utilize environments from the MuJoCo continuous control benchmark suite (first introduced by Todorov et al. [2012]), implemented via the Gymnasium interface:

- **Hopper-v5**: 2D one-legged agent learns to hop forward.
- **Ant-v5**: 3D quadruped robot learns to walk forward.
- **HalfCheetah-v5**: 2D two-legged agent learns to run forward.
- **Pusher-v5**: 3D arm learns to push an object to a goal.

Each agent receives reward signals based on task-specific objectives, control efficiency, and the novelty of chosen actions. All agents were trained for 1 million steps per environment across 10 random seeds.

Related Works

Prior research has explored combining evolutionary algorithms and DRL to address sample inefficiency and instability. Notably, Such et al. [2017] demonstrated that pure genetic algorithms can match DRL performance on high-dimensional tasks. Khadka and Tumer [2018] proposed ERL, a hybrid that integrates GA and DDPG via shared experience and policy blending. Pourchot and Sigaud [2018] combined the Cross-Entropy Method with TD3, achieving both stability and sample efficiency.

Maus et al. [2025] investigated the use of genetic algorithms to pre-seed the experience buffer to bootstrap useful experiential data early on when it is noisiest. While effective, their approach uses GAs as an auxiliary data generation tool rather than a method for direct policy weight initialization.

In contrast, our work focuses on direct GA-based policy initialization—warm-starting PPO and SAC agents using evolved network parameters, without modifying the learning algorithms or pre-populating experiential buffers. This enables a clean comparison against standard random initialization and isolates the impact of GA-derived weights on training dynamics. Our evaluation emphasizes reward variance and policy convergence (evaluation episode duration) in continuous control tasks, extending prior studies to newer MuJoCo environments and a more variance-centric analysis.

Methods

We implemented two GA variants for policy pre-training:

- **GA1 – Crossover + Mutation**: Uses binary mask crossover and Gaussian noise mutation. Fitness is defined as average episode reward.
- **GA2 – Mutation + Novelty**: Omits crossover and introduces larger Gaussian noise. Fitness combines reward and behavioral novelty via a weighted score. Novelty is estimated via average distance in a feature embedding space.

Each variant evolved populations of 20 neural networks for 20 generations. Networks were evaluated using a maximum of 1000 steps in per generation, so collectively the 20 generations of 20 networks each cumulatively account for $400,000$ environmental steps). The top-performing individual initialized the policy network of PPO or SAC and value networks and replay buffers were left randomly initialized or empty to match baselines for fairness.

Integration Strategy: For each experimental condition (baseline, GA1, GA2), we trained agents using 1 million environment steps per run across 10 random seeds. With the exception of the baseline non-GA runs, the actor network was initialized from the GA's top individual in the warm-start conditions, while the critic networks (and target networks in SAC) were left randomly initialized. Replay buffers were also left empty at the start of training. This setup allowed us to directly isolate the effect of policy weight initialization on downstream DRL performance without trying to disentangle the effect of pre-filled buffers of the effect of also pre-training non-policy networks.

The `cleanrl` package [Huang et al., 2021] was leveraged for the analysis with their default implementations PPO and SAC, with no changes to default hyperparameters, ensuring high reproducibility and consistency. All training runs, model artifacts, and metric visualizations were tracked using Weights & Biases (W&B), with the project registered publicly at: https://wandb.ai/ga-rl-final-project/projects. The logging framework captured episode-level rewards, policy entropy, duration rates, and seed-wise variation, enabling in-depth comparisons of intra- and inter-agent variability and reward-based success across all configurations. In total, the full experiment matrix comprised 4 environments $\times$ 2 DRL algorithms $\times$ 3 initialization strategies (baseline, GA1, GA2) $\times$ 10 random seeds, yielding 240 independent training runs. This volume was to allow statistical analysis of both performance and variability under each configuration, but due to computational and temporal constraints 10 seeds was chosen instead of a more preferable 30+ seeds.

Experimental Results & Analysis

Metrics

After training, each agent is trialed over 100 evaluation episodes. Our evaluation metrics are *episodic reward* and *duration rate*.

Episodic reward is the amount of reward collected by the agent throughout the evaluation episode. *Duration rate* is the ratio of the number of timesteps the agent remained healthy in the environment, divided by the maximum number of timesteps possible in the environment. Of principal importance is the standard error of our estimates of episodic reward; we aim to show lower standard error at comparable performance with generic algorithm pretraining. Duration rate serves as additional evidence for learned policy quality beyond accumulated reward; a stable policy should remain healthy for most or all of the episode.

Experimental Results

Table 1 summarizes performance outcomes across all environments, algorithms, and initialization strategies. We report the average episodic reward, average duration rate, and the average episodic reward standard error (SE) across 10 random seeds per configuration. Bold entries indicate the best-performing strategy within an environment-algorithm pair.

Analysis

Our experiments did not find strong evidence that GA-initialized weights improved performance or stability relative to standard random initialization. Consistent across the 4 MuJoCo environments and both DRL algorithms, baseline random weight initialization either matched or clearly outperformed GA-based initialization in most configurations, albeit with low significance and heavily-overlapping standard errors.

- In **Ant-v5** with SAC, baseline agents achieved higher mean rewards than both GA variants, but all three configurations had overlapping standard errors exceeding 300, making these

Env ID	Algo	Weights	Episodic Return Avg	Duration Rate Avg	SE(Return)
Ant-v5	PPO	BASELINE	17.9903	0.5622	1.6583
	PPO	GA1	17.9417	0.6029	1.3225
	PPO	**GA2**	**18.8686**	**0.6278**	1.0669
	SAC	**BASELINE**	**4587.5716**	**0.9252**	332.4050
	SAC	GA1	3865.0050	0.8483	401.9355
	SAC	GA2	4175.6116	0.9189	965.6828
HalfCheetah-v5	PPO	BASELINE	47.3023	1.0000	9.7371
	PPO	**GA1**	**48.4657**	**1.0000**	8.2989
	PPO	GA2	41.2704	1.0000	6.1439
	SAC	**BASELINE**	**9563.7627**	**1.0000**	332.4050
	SAC	GA1	8916.5836	1.0000	401.9355
	SAC	GA2	687.8516	1.0000	965.6828
Hopper-v5	PPO	BASELINE	1.7991	0.0374	0.4781
	PPO	**GA1**	**10.7885**	**0.1785**	7.3888
	PPO	GA2	1.8189	0.1641	0.4106
	SAC	**BASELINE**	**1354.5389**	**0.4020**	238.6120
	SAC	GA1	1168.2378	0.3481	187.2019
	SAC	GA2	292.6023	0.1090	138.1388
Pusher-v5	**PPO**	**BASELINE**	**-8.0891**	**1.0000**	0.4499
	PPO	GA1	-8.3616	1.0000	0.3392
	PPO	GA2	-8.5977	1.0000	0.4224
	SAC	BASELINE	-35.2236	1.0000	0.3361
	SAC	**GA1**	**-35.1303**	**1.0000**	0.1935
	SAC	GA2	-35.7843	1.0000	0.4009

Table 1: Performance Summary Across Algorithms and Initialization Strategies

differences statistically inconclusive.

- In **HalfCheetah-v5**, both SAC and PPO baselines again led in reward. The GA2 variant under SAC yielded significantly lower mean return (687 vs. 9500+) with wide variability, signaling a significantly large negative effect associated with the novelty-driven GA search.
- In **Hopper-v5**, PPO with GA1 initialization appeared to outperform the baseline in mean reward and duration, but the standard error was nearly four times the baseline value since one of the 10 runs appears to be an outlier, performing anomalously good, but with the performance differential still not very significant.
- In **Pusher-v5**, all configurations converged to similar negative reward values (PPO near -8, SAC near -35) with high duration rates, suggesting this environment was insensitive to initialization differences and perhaps predominately sensitive to the CleanRL default parameters which may not be suited to the environment. Coincidentally, in the breakout SAC

paper by Haarnoja et al. [2018], the Pusher environment is not included in their performance comparisons.

Taken together, the effect of genetic algorithm pretraining appears inconsistent, generally weak, and, on average, detrimental to reward optimization. Across nearly all environment–algorithm combinations, baseline agents achieved comparable or superior performance. The magnitude of standard errors in most configurations either exceeds or closely matches the differences in mean returns, indicating that these variations are not statistically significant.

Given the limited 10-seed per configuration limitation and to avoid the multiple comparison problem across 24 sets of configurations, we avoid formal statistical testing in favor of descriptive inference. Any observed improvements (e.g., GA2 in PPO Ant-v5) are likely attributable high effect variance or initialization noise rather than consistent benefits from GA pretraining.

Listed are several explanations we believe plausible or worthy of investigation for why GA-initialized agents may have underperformed:

1. **Learning rate mismatch:** A policy initialized with evolved weights may require a lower initial learning rate to avoid large destructive updates. CleanRL defaults do not account for warm-started policies and may prematurely overwrite useful pretraining signals.
2. **Critic–actor mismatch:** Our method resets the value network(s) and replay buffer after GA initialization. This can create a mismatch: the pretrained actor begins exploring in high-reward regions, but the critic starts from scratch, leading to unstable or misleading gradient updates.
3. **Lack of exploratory bootstrapping:** Baseline agents begin with random weights and experience a natural exploratory phase. In contrast, GA-initialized agents may begin in specialized policy regions but lack a diverse buffer to support generalization, effectively learning on biased or insufficient data early in training, often causing catastrophic errors and *failure to launch*.

Conclusion

While genetic algorithm pre-training offers an appealing strategy to inject robustness into DRL, our results suggest that naively GA warm-starting may not improve reward or stability. However, our methodology lays the groundwork for future research on more tightly integrated hybrid models that combine the exploratory power of GAs with the precision of DRL updates. Adjusting hyperparameters, incorporating replay buffer pre-filling from GA episodes are promising directions for future work.

References

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290v2*, 2018. URL https://arxiv.org/pdf/1801.01290.

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, and Jeff Braga. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms, 2021. URL https://arxiv.org/abs/2111.08819.

Ram Khadka and Kagan Tumer. Evolutionary reinforcement learning. In *Advances in Neural Information Processing Systems*, 2018.

Daniel Maus, Cynthia Tang, and Javier Ortiz. Leveraging genetic algorithms for efficient demonstration generation in reinforcement learning. *arXiv preprint arXiv:2507.00762*, 2025. URL https://arxiv.org/abs/2507.00762.

Nicolas Pourchot and Olivier Sigaud. Cem-rl: Combining evolutionary and gradient-based methods for policy search. *arXiv preprint arXiv:1810.01222*, 2018.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347v2*, 2017. URL https://arxiv.org/pdf/1707.06347.

Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*, 2017.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *IROS*, pages 5026–5033. IEEE, 2012. ISBN 978-1-4673-1737-5. URL http://dblp.uni-trier.de/db/conf/iros/iros2012.html#TodorovET12.

Appendix

A. Source Code (GitHub) and Training Results (Weights & Biases)

All training was synchronized with Weights & Biases, logging losses, rewards, and model checkpoints, including the final weights used for evaluation. Runs are organized into projects by algorithm, environment, and set of weights.

https://wandb.ai/ga-rl-final-project/projects

All code used in this analysis is available in a public GitHub repository. From this repository one can regenerate the entire results of this paper; see included README.md.

https://github.com/JamesCWilliams/deep_learning_developments_with_pytorch_final_project.

B. Sample Hyperparameters

- CleanRL Defaults - PPO: learning rate = 3e-4, clip range = 0.2, batch size = 64
- CleanRL Defaults - SAC: learning rate = 3e-4, buffer size = 1e6, batch size = 256
- GA1: pop size = 20, mutation std = 0.05, crossover probability = 0.5
- GA2: pop size = 20, mutation std = 0.10, crossover probability = 0, novelty method = nearest neighbors, novelty rank weight = 25%

C. GA Evaluation Pipeline

1. Initialize population with random weights.
2. Evaluate fitness using environment rollouts.
3. Select elites and apply crossover/mutation.
4. Repeat for 20 generations, then save top individual.

D. Visualizations of Evaluation Results

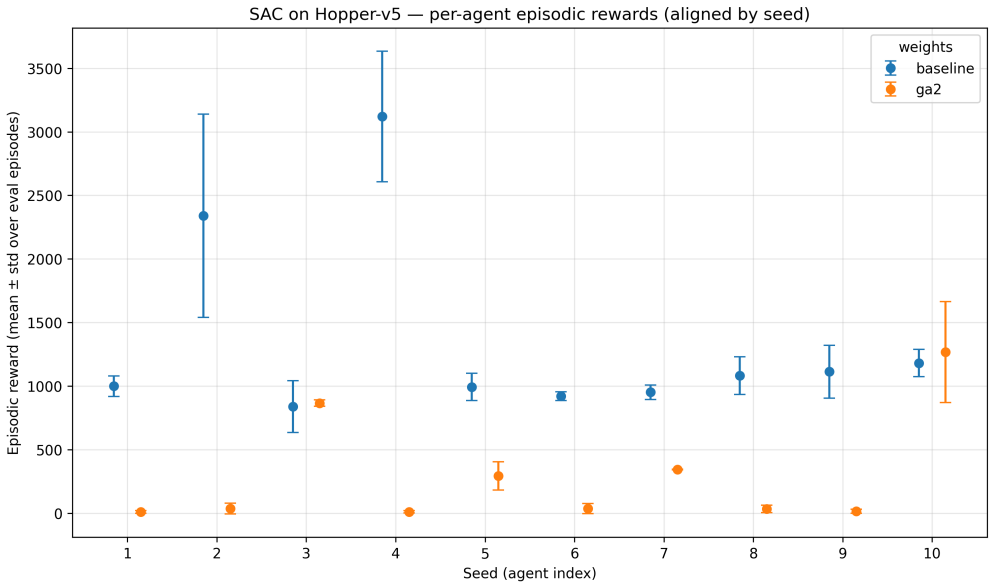

Figure 1: Per-agent episodic return means and standard deviations for SAC on Hopper-v5.

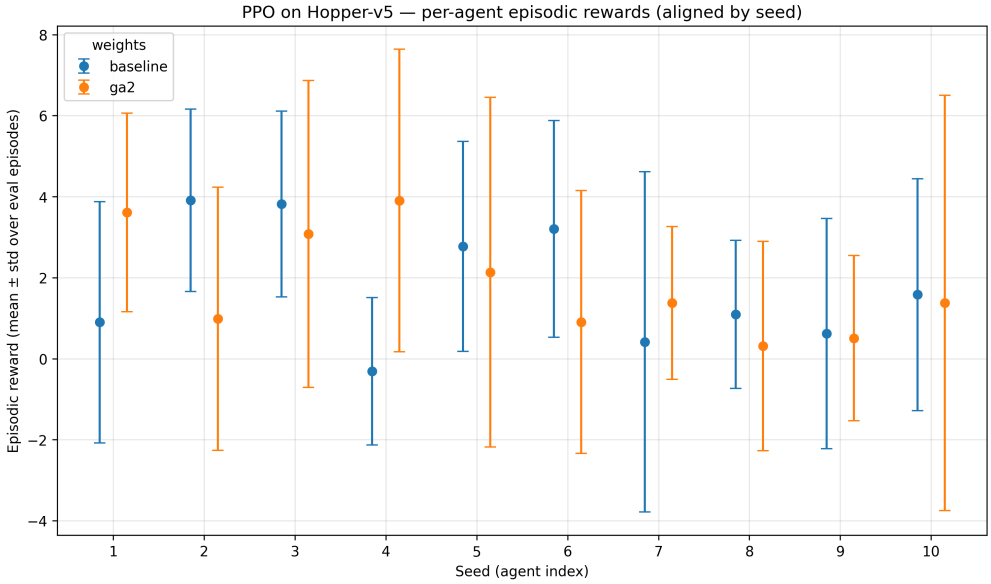

Figure 2: Per-agent episodic return means and standard deviations for PPO on Hopper-v5.

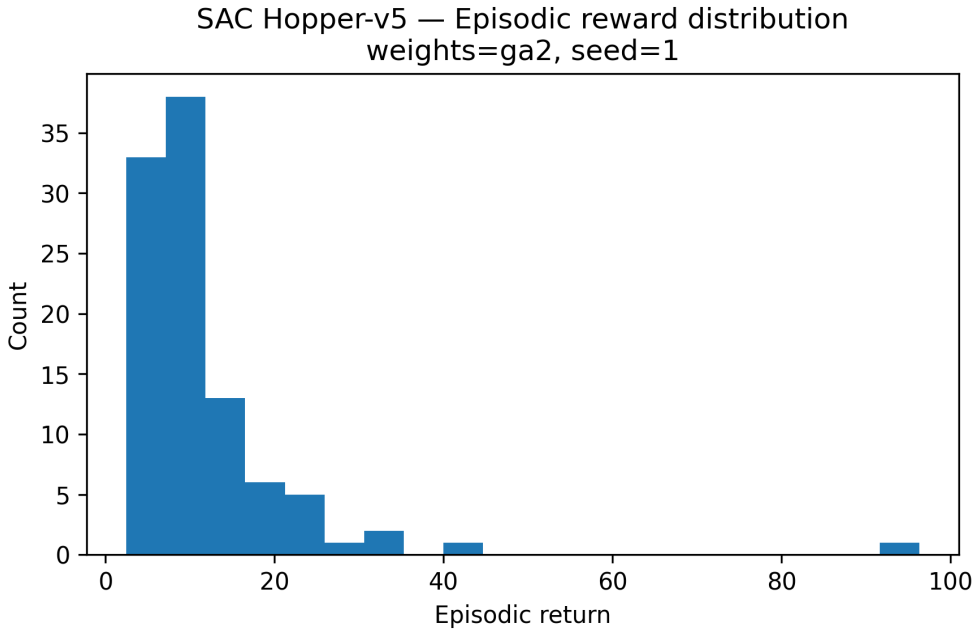

Figure 3: Histogram of episodic returns, SAC on Hopper-v5.

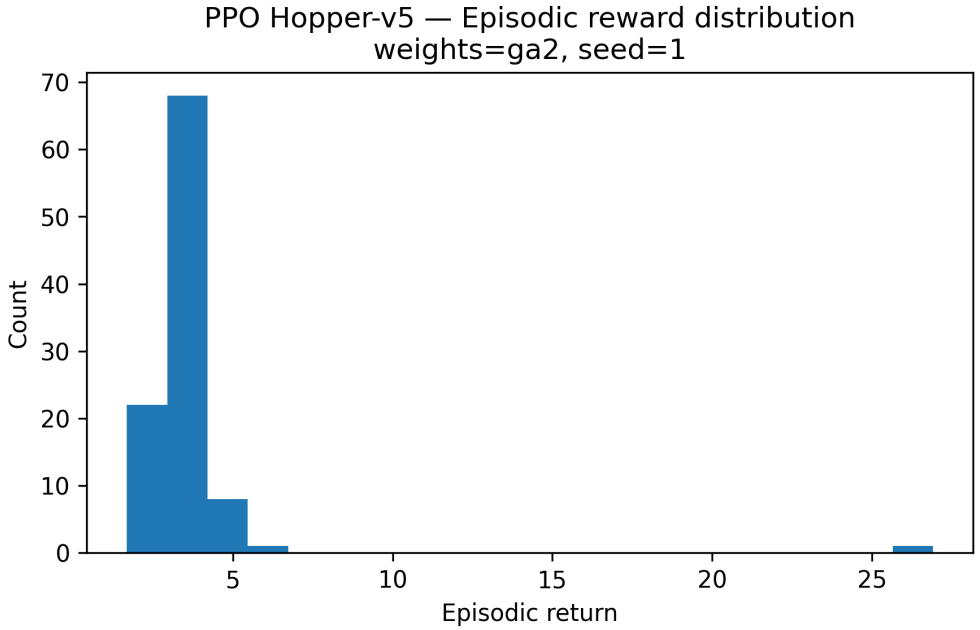

Figure 4: Histogram of episodic returns, PPO on Hopper-v5.

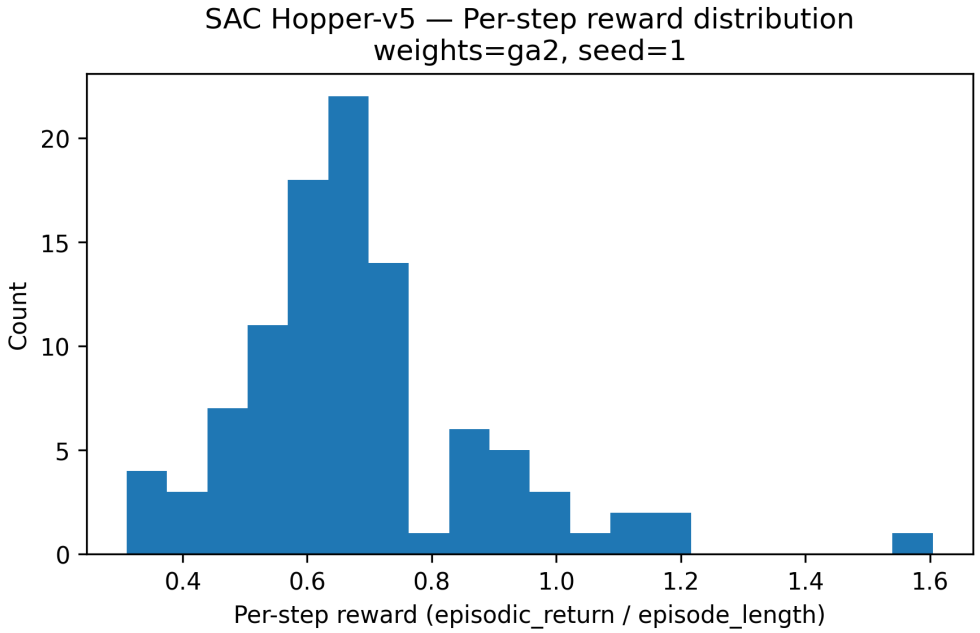

Figure 5: Histogram of episodic returns, SAC on Hopper-v5, normalized by episode length.

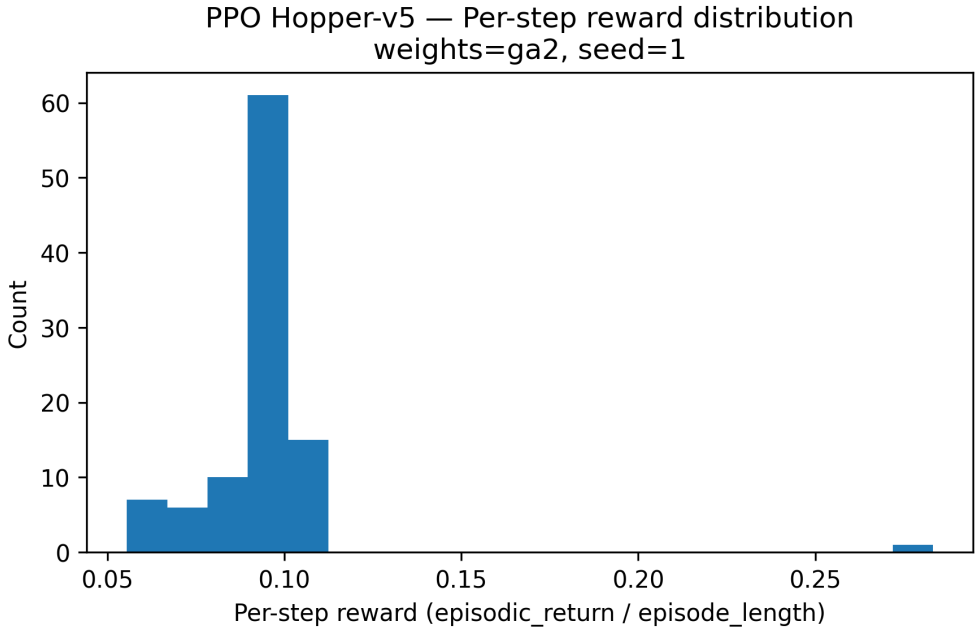

Figure 6: Histogram of episodic returns, PPO on Hopper-v5, normalized by episode length.