Reflective Prompted Policy Optimization: Trajectory-Grounded Revision and Salience Bias

Rahaf Abu Hara, Vaibbhav Murarri, Claudio Zito
Laboratory of AI and Robotics Research (LAIRR)
Heriot-Watt University
Dubai, United Arab Emirates
{rha4001,vm81,C.Zito}@hw.ac.uk

Abstract

Existing LLM-based policy optimizers see only scalar rewards: that a policy scored 0.45, but not whether the agent got stuck in a loop, fell into a hole on the third step, or performed well on 19 out of 20 rollouts and failed catastrophically on one. We propose Reflective Prompted Policy Optimization (R2PO), a two-stage LLM framework for policy search over compact policy classes that augments scalar reward feedback with trajectory-level behavioral evidence. A *Search-LLM* acts as a global policy optimizer and proposes candidate policy parameters; the environment executes them; a *Critic-LLM* then inspects the resulting rollouts and proposes targeted parameter revisions grounded in observed states, actions, and rewards. Across ten environments, ablations show R2PO's gains arise from a design that explicitly separates global search from behavior-grounded revision and uses selection to filter high-variance edits. We further identify a dominant failure mode, *salience bias*. When presented with multiple rollouts, the Critic-LLM fixates on improving a single failure even when most trajectories succeed. In a three-trajectory variant, where the Critic-LLM is shown the best, worst, and median rollout from each evaluation, this behavior explains 76.6% of regressions on CartPole. R2PO mitigates this by reasoning over aggregate rollout statistics, median-trajectory selection, and a revision rule. Using a relatively small open-weight 20B-parameter model, R2PO achieves the highest mean best reward across all ten environments compared with deep RL and prior LLM-based methods, while reaching near-optimal performance substantially earlier in training (e.g., near-maximum CartPole reward within $\sim$500 episodes), and training far more stably than both deep RL and prior LLM-based methods. Together, these results show that treating trajectories as first-class in-context evidence, rather than background artifacts reduced to scalar returns, fundamentally changes how even comparatively small LLMs can search over policy spaces, enabling them to learn faster, diagnose more precisely, and reliably improve external controllers rather than tune them by trial and error.

1 Introduction

Reinforcement learning has made substantial progress on policy optimization, but standard algorithms require careful hyperparameter tuning and millions of environment interactions to converge, with performance sensitive to implementation details. Recent work has shown that large language models can serve as black-box optimizers for policy search, consuming histories of parameter-reward pairs and proposing new candidate policies through in-context learning [28, 29]. These methods treat the LLM as an optimizer over the policy parameter space, requiring no task-specific algorithmic design.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

However, existing LLM-based policy optimizers operate on scalar reward feedback alone. The optimizer sees *that* a policy scored 0.45, but not *why*: whether the agent got stuck in a loop, fell into a hole on the third step, or performed well on 19 out of 20 rollouts but catastrophically failed on one. Scalar feedback provides no diagnostic signal about what went wrong, limiting the optimizer to trial-and-error tuning rather than targeted correction.

We introduce **Reflective Prompted Policy Optimization (R2PO)**, a two-stage LLM framework for policy search over compact policy classes (linear policies for continuous control and tabular policies for discrete tasks). A *Search-LLM* acts as a global policy optimizer: given the history of evaluated policies and their scalar rewards, it proposes a candidate policy parameter vector. The candidate is executed in the environment, producing rollout trajectories that capture states, actions, and rewards. A *Critic-LLM* then inspects this evidence and proposes a targeted parameter revision. Finally, a selection step compares the original and revised candidates and keeps whichever achieves the higher reward, limiting damage from over-corrections or LLM hallucinations.

Trajectory-grounded diagnosis is a powerful tool that enables better decisions than scalar-only feedback, but it can also cause sharp regressions when the trajectories used for diagnosis are unrepresentative. We identify a recurring failure pattern, *salience bias*: when the Critic-LLM is shown multiple trajectories from the same policy, it treats the worst-case trajectory as representative, even when the policy is working well. We mitigate salience bias by feeding the Critic-LLM a more representative view of policy behavior: (i) aggregate rollout statistics that summarize overall performance, (ii) a median trajectory that reflects typical rather than worst-case behavior, and (iii) a revision rule that preserves policies that already perform well.

Our contributions are:

- **R2PO**, a two-stage LLM policy-optimization framework that combines scalar reward search with trajectory-grounded behavioral diagnosis, treating rollouts as first-class in-context evidence.
- **A systematic ablation** across ten environments over search budget, prompt design, two-stage architecture, selection, and trajectory evidence, showing that R2PO's gains cannot be explained by more LLM calls or richer prompts alone, but require the combination of trajectory-grounded revision, two-stage design, and keep-best selection.
- **A mechanistic analysis** identifying and quantifying a dominant failure mode of trajectory-grounded LLM revision, *salience bias*, in which the Critic-LLM overweights vivid worst-case rollouts when multiple trajectories are available, accounting for 76.6% of regressions in a three-trajectory setting on CartPole.
- **Empirical validation** using a 20B open-weight LLM: compared with deep RL and prior LLM-based methods, R2PO achieves the highest mean reward on 9 of 10 environments and is statistically tied on the 10th, matches or exceeds these baselines on mean best reward, reaches near-optimal performance substantially earlier in training, and trains more stably. We further evaluate R2PO against classical black-box optimizers and on two simulated robotic tasks involving a 14-parameter decision policy and a structured 70-parameter motor policy.[1]

2 Related Work

Recent work integrates LLMs into RL pipelines in two main ways. One line uses LLMs to shape components of the RL loop, such as reward functions, action selection, or auxiliary policy guidance, while leaving optimization to conventional RL algorithms [11, 13, 6, 27, 30]. In these approaches, the LLM acts as an auxiliary module rather than as the optimizer itself.

A second line places the LLM at the center of optimization. OPRO [28] uses LLMs to optimize arbitrary objective functions through iterative prompting; ProPS [29] extends this to RL by consuming parameter-reward histories and proposing policy updates directly. Our work operates in this regime, but differs fundamentally: rather than optimizing from scalar feedback alone, we introduce a second LLM stage that reasons over the proposed policy's actual rollout behavior, enabling targeted diagnosis of failure modes that scalar rewards cannot reveal.

Self-improvement and reflection methods for LLMs are also related. Self-Refine [14] performs iterative generate-critique-revise loops for single outputs; Reflexion [23] extends this to agents by

[1] Code is available at: `https://github.com/HWUDLabAIRoboticsResearch/R2PO`

converting task feedback into verbal reflections stored in episodic memory. In Reflexion, the LLM *is* the policy and iteratively retries the same task using verbal reflections from prior attempts. In R2PO, the LLM is an *optimizer* of separate policy parameters: the Critic-LLM diagnoses an external policy's rollout behavior and produces a concrete parameter vector evaluated against the original.

A recent direction uses trajectories as supervision for LLM *training* via gradient updates: TreeRL [10] adds on-policy tree search with process supervision, and TSR [3] constructs higher-quality training trajectories through lightweight tree-style search. R2PO instead uses rollout trajectories as *in-context evidence* to revise external policies, with no gradient computation or fine-tuning.

3 Methodology

3.1 Problem setting

We consider episodic policy search with parameterized policies π_θ and the objective of maximizing expected return $J(\theta) = \mathbb{E}[R(\tau)]$, where τ is a trajectory generated by π_θ. We focus on compact policy classes whose parameter vectors fit within an LLM context window: linear policies $\pi_\theta(s) = \theta^\top \phi(s)$ for continuous-state tasks ($\arg\max$ over action logits for discrete actions) and tabular policies $\pi_\theta(s) = \theta_s$, with $\theta_s \in \mathcal{A}$, for discrete-state tasks. Extending R2PO to high-dimensional neural policies is an important direction for future work (Section 5).

Given a parameter vector θ, we evaluate it by running K independent environment rollouts and averaging their returns. We denote this evaluation

$$\textsc{Eval}(\theta) \to (\bar{R}, \mathcal{T}),$$

where $\bar{R}$ is the mean return and $\mathcal{T} = \{\tau_1, \ldots, \tau_K\}$ is the set of recorded trajectories.

3.2 The R2PO framework

R2PO uses two LLMs in distinct roles (Figure 1). The Search-LLM is the *global policy optimizer*: given the history of previously evaluated policies and their scalar mean rewards, it proposes where to move next in the parameter space. Scalar history suffices at this global level, where candidates are compared on a uniform ranking signal; trajectory-grounded reasoning is reserved for the local revision step. The Critic-LLM is the *trajectory-based reviser*: given a calibrated summary of the behavior observed when the Search-LLM's proposal was executed, it inspects what went wrong and proposes a targeted revision. In practice, both roles use the *same* underlying 20B model, but via two independent calls with distinct prompts tailored to their respective roles.

Stage 1: Search-LLM proposal. At iteration t, the Search-LLM is prompted with the optimization task and a reward-only history $H_t = \{(\theta_i, \bar{R}_i)\}_{i=1}^{t-1}$, and proposes an initial parameter vector $\theta_t^{(0)}$, which the environment evaluates as $(\bar{R}_t^{(0)}, \mathcal{T}_t^{(0)})$.

Stage 2: Critic-LLM reflection. The Critic-LLM is prompted with a natural-language environment description, the proposed parameters $\theta_t^{(0)}$, the same reward-only history H_t, and a calibrated trajectory evidence package built from the K rollouts in $\mathcal{T}_t^{(0)}$. From this evidence, it proposes a revised parameter vector $\theta_t^{(1)}$. The evidence package has three components.

(i) Median trajectory. From the K rollouts, we select the rollout whose return is closest to the median of the K returns (interpolated when K is even). This provides a concrete example of typical policy behavior, avoiding both best-case and worst-case outliers.

(ii) Aggregate rollout statistics. A structured statistical report computed over all K rollouts: reward mean, minimum, and maximum; episode length mean, minimum, and maximum; and success and failure rates. These aggregates convey how often the policy fails and by how much without placing a specific failed execution trace in front of the Critic-LLM.

(iii) Revision rule. We add an explicit abstention instruction to the Critic-LLM's prompt, controlled by a threshold τ_c. If the initial mean reward $\bar{R}_t^{(0)}$ exceeds τ_c and the rollout statistics do not indicate a consistent failure pattern, the Critic-LLM is instructed to keep the parameters unchanged or make only

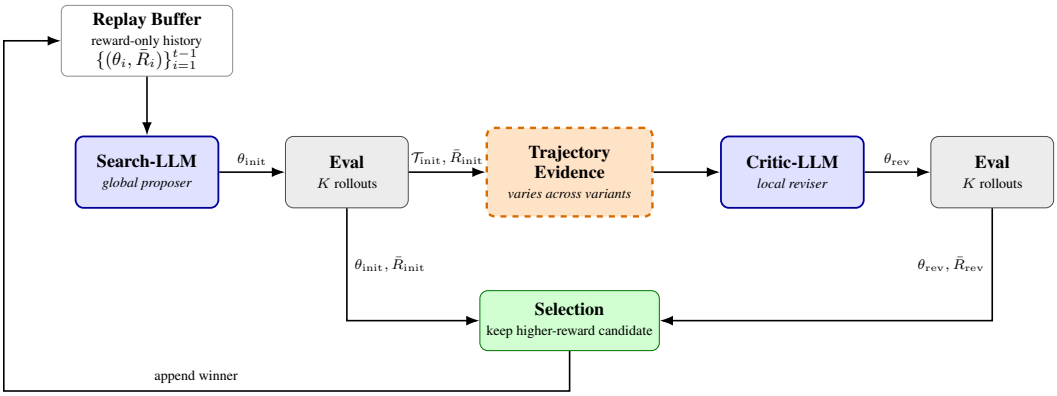


Figure 1: **The two-stage R2PO framework.** The Search-LLM proposes candidate parameters θ_{init} conditioned on a reward-only replay history $\{(\theta_i, \bar{R}_i)\}_{i=1}^{t-1}$ of previously evaluated parameters and their mean rewards. The environment evaluates θ_{init} over K rollouts, returning the mean reward and rollout trajectories from which *Trajectory Evidence* (orange, dashed) is constructed. The Critic-LLM uses this evidence to propose a revised vector θ_{rev}, which is re-evaluated; a selection step retains whichever candidate achieves the higher reward before appending it to the replay buffer. Trajectory Evidence is the central design choice we study: in R2PO, it consists of the median rollout, aggregate rollout statistics, and a revision rule applied when the policy is already strong.

a very small, strongly justified edit. The threshold τ_c is set per environment based on the observable reward scale (see Appendix F).

Selection. We evaluate the revised parameters and keep whichever candidate performs better:

$$\theta_t = \begin{cases} \theta_t^{(1)} & \text{if } \bar{R}_t^{(1)} \geq \bar{R}_t^{(0)}, \\ \theta_t^{(0)} & \text{otherwise.} \end{cases} \tag{1}$$

The selected parameters and their mean reward are appended to the history for subsequent iterations. Algorithm 1 in Appendix E summarizes the full R2PO loop.

4 Experiments

4.1 Experimental setup

We evaluate on ten environments spanning discrete and continuous action spaces, as well as stochastic and deterministic dynamics: CartPole-v1, FrozenLake-v1, MountainCar-v0, MountainCarContinuous-v0, InvertedPendulum-v5, InvertedDoublePendulum-v5, and Swimmer-v5 [26, 25], and three custom environments (Maze, Nim, and Pong). All LLM calls use `gpt-oss:20b-cloud`, an Ollama deployment of the OpenAI gpt-oss model family, with cloud-hosted inference [18, 17]. SB3 baseline hyperparameters are taken from the official Stable-Baselines3 library [20], using RL Baselines3 Zoo [19] presets where available and library defaults otherwise. The revision threshold τ_c is set near each environment's strongest empirically achievable reward (Appendix F). Appendix B shows R2PO is robust to $\pm 50\%$ threshold variation and remains competitive without it.

We compare three method tiers. *Baselines*: ProPS, ProPS+ [29], the best-performing Stable-Baselines3 (SB3) [20] algorithm per environment selected from PPO [22], TRPO [21], SAC [5], DQN [15], A2C [16], DDPG [12], and TD3 [4], and three classical black-box optimizers: Random Search [1], CMA-ES [7], and Bayesian Optimization (BayesOpt) [24]. *Decomposition ablations*: PureSearch, ActorSecondPass, CriticOnly, and AlwaysCritic, each isolating one factor that could explain R2PO's gains (Section 4.3). *Trajectory-evidence variants*: R2PO, RepTraj , and ThreeTraj, each instantiating a different construction of trajectory evidence for the Critic-LLM (Section 4.3).

All methods are evaluated over 10 independent runs with $K{=}20$ rollouts per evaluation. Each LLM-based method receives a matched budget of 200 LLM calls and 4,000 episodes per run: two-call methods (R2PO, RepTraj, ThreeTraj, AlwaysCritic, ActorSecondPass, PureSearch) run for

$T{=}100$ iterations with 2 LLM calls and $2K{=}40$ rollouts per iteration; single-call methods (ProPS, ProPS+, CriticOnly) run for $2T{=}200$ iterations with 1 LLM call and K rollouts per iteration. SB3 baselines train on 4,000 episodes per run. For the comparison with classical black-box optimizers, we match the number of candidate-policy evaluations across R2PO, Random Search, CMA-ES, and BayesOpt within each environment. Since CMA-ES updates its search distribution after complete generations, we use the largest complete-generation budget not exceeding 200 evaluations, yielding $N \in \{192, 196, 200\}$. CMA-ES uses the default population size in `pycma` [8], while BayesOpt uses `scikit-optimize` [9] with a Gaussian-process surrogate and Expected Improvement acquisition. We report two metrics, both averaged over 10 runs with $\pm$ standard deviation: *mean reward* (the average reward across all iterations) and *mean best reward* (mean of the best reward achieved during training).

4.2 Main results

Table 2 shows that the best policies found by R2PO match or exceed the baselines reported in the table across all 10 environments, reaching the maximum possible return on CartPole (500), InvertedPendulum (1000), Pong (3.0), and Nim (1.0). R2PO further achieves the highest mean reward across 9 of 10 environments among the methods in Table 1; the only exception is MountainCarContinuous, where SAC slightly leads (82.33 ± 2.57 vs our 81.61 ± 5.18). Compared to ProPS+, R2PO performs better on all 10 environments, often by large margins, and Appendix A shows that these gains are statistically significant.

Beyond peak performance, R2PO maintains strong policies once they are found: its mean rewards remain close to its best rewards (e.g., CartPole $\sim$475/500; InvertedPendulum $\sim$756/1000), indicating stable training rather than isolated spikes (Appendix C). Prior LLM-based methods are less stable (e.g., ProPS+ on InvertedPendulum reaches a best of $\sim$658 but averages only $\sim$310, and on MountainCarContinuous reaches $\sim$99 best vs. $\sim$18 mean). R2PO also learns faster, achieving near-optimal CartPole performance within $\sim$500 episodes (vs. $\sim$4,000 for ProPS+ and SB3) and surpassing ProPS, ProPS+, and the best SB3 baseline on Swimmer within $\sim$1,000 episodes. Figure 2 shows learning curves on four representative environments; full curves for all ten are in Appendix G.

Table 3 shows that R2PO achieves the highest or tied-highest mean best reward on 9 of 10 environments, exceeding Random Search on all ten and both CMA-ES and BayesOpt on eight, with a tie on CartPole. BayesOpt achieves the highest mean best reward on Swimmer.

We further evaluate R2PO on two simulated robotic tasks: dynamic ASV debris collection with a 14-parameter policy and robotic table tennis with a structured 70-parameter motor policy. R2PO achieves the highest observed held-out mean return on the ASV task and matches ProPS+ in table-tennis target success, reaching the success criterion with fewer training rollouts (180 vs. 980). Full results are provided in Appendices M and N.

Finally, Appendix D reports an ablation-style comparison against the published ProPS results [29], which were obtained with *double* the interaction budget (8,000 episodes) and a larger proprietary model (GPT-4o). Even under these conditions, R2PO—trained with 4,000 episodes and an open-weight GPT-oss:20B model—still matches or exceeds ProPS+ and the best SB3 baseline on 9 of 10 environments.

4.3 Ablation study and decomposition

R2PO combines (i) a two-stage LLM pipeline, (ii) trajectory-grounded behavioral evidence, (iii) a reward-only replay history, (iv) a revision rule, and (v) a keep-best selection mechanism. We ask two questions: *(1) which components are actually necessary for the observed gains?* and *(2) given a two-stage architecture, how should rollout evidence be summarized for the Critic-LLM?* We address (1) using architectural ablations (Section 4.3.1) and (2) using trajectory-evidence variants that differ only in what the Critic-LLM sees (Section 4.3.2). Table 4 reports mean reward across all ten environments; mean best reward (Appendix H) shows the same overall pattern.

4.3.1 Decomposition: are all components of R2PO necessary?

We design four ablation variants, each removing or altering a single aspect of R2PO.

Table 1: Mean reward ($\pm$ standard deviation) across 10 independent runs, averaged across all training iterations. **Best** per environment is highlighted. R2PO achieves the highest mean reward on 9 of 10 environments and is statistically tied with SAC on MountainCarContinuous (81.61 ± 5.18 vs. 82.33 ± 2.57).

Environment	ProPS	ProPS+	Best SB3	R2PO
Nim	-0.59 ± 0.21	-0.25 ± 0.07	0.01 ± 0.32 (A2C)	$\mathbf{0.61 \pm 0.03}$
Pong	0.74 ± 0.63	1.22 ± 0.58	1.02 ± 0.74 (PPO)	$\mathbf{2.51 \pm 0.21}$
Swimmer	89.22 ± 48.68	162.05 ± 66.07	44.60 ± 7.30 (TRPO)	$\mathbf{260.35 \pm 36.05}$
MountainCarCon	-23.45 ± 41.65	17.90 ± 37.48	$\mathbf{82.33 \pm 2.57}$ (SAC)	81.61 ± 5.18
MountainCar	-199.31 ± 2.18	-195.81 ± 3.73	-199.99 ± 0.02 (DQN)	$\mathbf{-147.84 \pm 7.11}$
InvDblPend	79.44 ± 16.50	86.71 ± 17.70	86.04 ± 0.33 (TRPO)	$\mathbf{158.51 \pm 68.41}$
InvPendulum	234.14 ± 169.76	309.52 ± 247.92	24.35 ± 0.13 (TRPO)	$\mathbf{756.08 \pm 154.50}$
FrozenLake	0.05 ± 0.05	0.37 ± 0.06	0.02 ± 0.02 (TRPO)	$\mathbf{0.62 \pm 0.07}$
CartPole	258.09 ± 138.58	253.06 ± 133.35	216.92 ± 63.34 (TRPO)	$\mathbf{474.67 \pm 16.90}$
Maze	-1.03 ± 0.18	0.76 ± 0.05	$\mathbf{0.83 \pm 0.13}$ (A2C)	$\mathbf{0.83 \pm 0.06}$

Table 2: Mean best reward ($\pm$ standard deviation) across 10 independent runs. **Best** per environment is highlighted. R2PO achieves the strongest peak performance on all environments.

Environment	ProPS	ProPS+	Best SB3	R2PO
Nim	0.75 ± 0.30	$\mathbf{1.00 \pm 0.00}$	0.88 ± 0.27 (A2C)	$\mathbf{1.00 \pm 0.00}$
Pong	2.15 ± 0.95	2.78 ± 0.41	2.33 ± 0.78 (PPO)	$\mathbf{3.00 \pm 0.00}$
Swimmer	208.52 ± 68.38	274.86 ± 38.81	67.89 ± 20.92 (TRPO)	$\mathbf{294.57 \pm 43.79}$
MountainCarCon	75.37 ± 30.57	98.70 ± 0.50	94.81 ± 0.42 (SAC)	$\mathbf{98.75 \pm 0.44}$
MountainCar	-191.84 ± 25.80	-150.21 ± 26.62	-197.47 ± 3.66 (DQN)	$\mathbf{-111.04 \pm 3.92}$
InvDblPend	112.18 ± 20.94	128.81 ± 54.51	98.25 ± 2.56 (TRPO)	$\mathbf{254.04 \pm 232.39}$
InvPendulum	649.91 ± 432.17	657.88 ± 444.42	28.49 ± 0.71 (TRPO)	$\mathbf{1000.00 \pm 0.00}$
FrozenLake	0.24 ± 0.14	0.90 ± 0.04	0.12 ± 0.12 (TRPO)	$\mathbf{0.93 \pm 0.05}$
CartPole	427.74 ± 143.04	396.21 ± 155.21	490.76 ± 29.22 (TRPO)	$\mathbf{500.00 \pm 0.00}$
Maze	-0.70 ± 0.88	$\mathbf{0.97 \pm 0.00}$	$\mathbf{0.97 \pm 0.00}$ (A2C)	$\mathbf{0.97 \pm 0.00}$

Is it just extra budget? PureSearch draws two proposals from the same Search-LLM context and keeps the better one, doubling the search budget without structural change. **ActorSecondPass** runs the scalar-only optimizer twice in sequence: the second call sees the first call's result and reward but no trajectory information. Both variants test whether R2PO's gains reduce to additional LLM calls or a generic sequential structure. Both are consistently among the weakest methods, failing systematically on Maze and Nim. Extra budget, whether parallel or sequential, does not explain R2PO's gains.

Is it just the richer prompt? CriticOnly replaces the two-stage loop with a single rich Critic-style prompt that internally performs draft–critique–revise within one LLM call, receiving scalar history and environment description but never actual rollouts. This isolates prompt design from the effect of having a separate reviser that reasons over realized behavior. CriticOnly is the strongest single-factor variant, yet R2PO outperforms it on all 10 environments, with large gaps on Swimmer ($\sim$260 vs. $\sim$55) and InvertedPendulum ($\sim$756 vs. $\sim$127). A richer prompt helps but cannot substitute for a two-stage design in which the reviser accesses execution behavior.

Is selection necessary? AlwaysCritic is identical to the two-stage variant RepTraj (Section 4.3.2, which uses a single representative rollout as trajectory evidence), except that it always commits the Critic-LLM's revision without selection. RepTraj outperforms AlwaysCritic on all 10 environments, often by large margins (InvertedPendulum: $\sim$380 vs. $\sim$141; MountainCarContinuous: ~ -94 vs. ~ -131). With architecture and evidence held constant, committing all revisions unconditionally introduces large variance and frequent regressions. We investigate why trajectory evidence fails to help when revisions are unconditional in Section 4.4, showing that it reflects high-variance, trajectory-conditioned edits often based on unrepresentative rollouts.

Summary. No single ablation recovers R2PO's performance. Extra LLM calls (PureSearch, ActorSecondPass), richer prompts alone (CriticOnly), and two-stage reasoning without selection (AlwaysCritic) are each insufficient. R2PO's gains arise from the *combination* of (i) a two-stage architecture separat-

Table 3: Mean best reward ($\pm$ standard deviation) across 10 independent runs. All methods, including R2PO, use matched environment-specific candidate-policy evaluation budgets ($N \in \{192, 196, 200\}$; see Section 4.1). **Best** per environment is highlighted. R2PO achieves the highest or tied-highest reward on 9 of 10 environments.

Environment	R2PO	Random Search	CMA-ES	BayesOpt
Nim	$\mathbf{1.00 \pm 0.00}$	-0.13 ± 0.24	0.20 ± 0.49	0.41 ± 0.19
Pong	$\mathbf{3.00 \pm 0.00}$	2.56 ± 0.65	2.65 ± 0.51	2.84 ± 0.33
Swimmer	294.32 ± 43.96	209.02 ± 29.69	297.16 ± 94.98	$\mathbf{305.93 \pm 74.57}$
MountainCarCon	$\mathbf{98.75 \pm 0.44}$	58.75 ± 29.81	44.75 ± 49.04	82.60 ± 29.36
MountainCar	$\mathbf{-111.03 \pm 3.92}$	-199.98 ± 0.06	-200.00 ± 0.00	-177.50 ± 27.30
InvDblPend	$\mathbf{254.04 \pm 232.39}$	124.58 ± 23.72	140.44 ± 30.63	142.54 ± 16.21
InvPendulum	$\mathbf{1000.00 \pm 0.00}$	84.22 ± 42.34	732.46 ± 435.95	711.23 ± 357.13
FrozenLake	$\mathbf{0.92 \pm 0.05}$	0.36 ± 0.17	0.48 ± 0.23	0.48 ± 0.23
CartPole	$\mathbf{500.00 \pm 0.00}$	292.17 ± 137.27	$\mathbf{500.00 \pm 0.00}$	$\mathbf{500.00 \pm 0.00}$
Maze	$\mathbf{0.97 \pm 0.00}$	0.34 ± 1.00	-0.70 ± 0.88	-0.28 ± 1.07

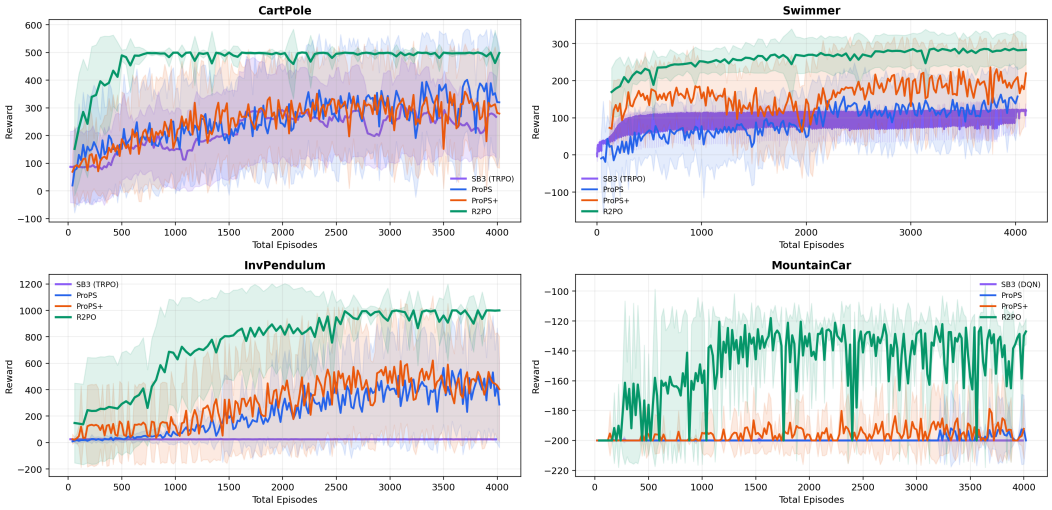


Figure 2: Learning curves (mean $\pm$ standard deviation over 10 independent runs) for R2PO, ProPS, ProPS+, and the best SB3 baseline on four representative environments. R2PO reaches strong performance earlier and maintains it more consistently than other baselines. Curves for the remaining six environments are in Appendix G and show the same overall pattern.

ing global proposal from local revision, (ii) trajectory-grounded reasoning in the reviser, and (iii) a keep-best selection rule that filters destructive edits.

4.3.2 Trajectory representation: how should the Critic-LLM see rollouts?

With the two-stage architecture and selection rule fixed, the remaining design question is how to summarize K rollouts into a compact representation that fits in context and remains interpretable to the Critic-LLM. Feeding all K trajectories as raw traces would exceed typical context limits and, in our pilots, led to incoherent edits; we therefore focus on compact summaries that preserve the key behavioral patterns. Providing only the scalar mean discards the behavioral information that motivates R2PO. We compare three structured representations, which differ only in how they summarize $\mathcal{T}_t^{(0)}$.

RepTraj: single representative trajectory. RepTraj selects the rollout whose return is closest to the mean across the K rollouts and passes it as a raw trajectory trace, together with the full reward and trajectory history. The Critic-LLM is always asked to revise, with no revision rule. This tests the simplest use of trajectory evidence: one approximately typical example.

Table 4: Mean reward ($\pm$ standard deviation) across 10 independent runs for R2PO and the ablation variants, averaged across all training iterations. **Best** per environment is highlighted. BestBaseline denotes the best among ProPS, ProPS+, and all SB3 algorithms. R2PO achieves the strongest average performance on most environments, while ThreeTraj wins by a small margin on 2 environments (Nim and FrozenLake, both discrete stochastic).

Environment	BestBaseline	PureSearch	ActorSecondPass	CriticOnly	AlwaysCritic	RepTraj	ThreeTraj	R2PO
Nim	0.01 ± 0.32 (A2C)	-0.36 ± 0.22	-0.40 ± 0.33	0.16 ± 0.05	0.07 ± 0.20	0.52 ± 0.13	$\mathbf{0.63 \pm 0.10}$	0.61 ± 0.03
Pong	1.22 ± 0.58 (ProPS+)	0.86 ± 0.49	1.03 ± 0.82	2.06 ± 0.58	1.86 ± 0.55	2.20 ± 0.41	1.83 ± 0.79	$\mathbf{2.51 \pm 0.21}$
Swimmer	162.05 ± 66.07 (ProPS+)	131.48 ± 78.68	111.89 ± 96.37	55.19 ± 58.55	111.45 ± 54.50	170.04 ± 68.11	109.56 ± 75.57	$\mathbf{260.35 \pm 36.05}$
MountainCarCon	$\mathbf{82.33 \pm 2.57}$ (SAC)	-21.59 ± 73.21	-124.66 ± 218.47	-85.08 ± 150.64	-130.71 ± 179.28	-93.85 ± 138.50	-29.88 ± 54.38	81.61 ± 5.18
MountainCar	-195.81 ± 3.73 (ProPS+)	-186.38 ± 20.21	-195.73 ± 13.50	-187.29 ± 7.87	-188.07 ± 9.73	-169.98 ± 20.36	-162.48 ± 19.64	$\mathbf{-147.84 \pm 7.11}$
InvDblPend	86.71 ± 17.70 (ProPS+)	86.90 ± 25.54	88.17 ± 8.98	128.87 ± 32.72	99.11 ± 21.14	132.96 ± 47.18	100.07 ± 24.90	$\mathbf{158.51 \pm 68.41}$
InvPendulum	309.52 ± 247.92 (ProPS+)	466.08 ± 357.96	480.08 ± 324.97	126.63 ± 123.89	141.41 ± 228.65	380.19 ± 213.27	194.25 ± 235.22	$\mathbf{756.08 \pm 154.50}$
FrozenLake	0.37 ± 0.06 (ProPS+)	0.08 ± 0.06	0.07 ± 0.04	0.31 ± 0.10	0.31 ± 0.08	0.47 ± 0.20	$\mathbf{0.63 \pm 0.06}$	0.62 ± 0.07
CartPole	258.09 ± 138.58 (ProPS)	337.72 ± 156.62	305.04 ± 123.88	418.23 ± 45.36	377.65 ± 85.88	416.15 ± 88.71	432.08 ± 59.56	$\mathbf{474.67 \pm 16.90}$
Maze	0.83 ± 0.13 (A2C)	-1.10 ± 0.03	-1.01 ± 0.33	0.68 ± 0.10	0.74 ± 0.07	0.77 ± 0.06	0.82 ± 0.12	$\mathbf{0.83 \pm 0.06}$

ThreeTraj: best, worst, and median trajectories. ThreeTraj aims to improve representativeness by exposing the Critic-LLM to the range of behaviors observed: it passes the highest-, lowest-, and median-reward trajectories from the K rollouts, with reward-only history H_t. The Critic-LLM is again always asked to revise. This tests whether richer within-policy evidence yields better corrections.

R2PO: median trajectory + aggregate statistics + Revision rule. R2PO shows the Critic-LLM (i) the median-reward trajectory, (ii) aggregate statistics over K rollouts (reward mean/min/max, length mean/min/max, success/failure counts), and (iii) a revision instruction (threshold τ_c) that preserves the policy when performance is already strong. Unlike RepTraj, R2PO does not show extreme rollouts; unlike ThreeTraj, it communicates failure frequency numerically rather than by foregrounding a worst-case trace.

Empirical comparison. As Tables 4 and 12 show, R2PO outperforms RepTraj on all 10 environments in mean reward. ThreeTraj achieves slightly higher mean reward than R2PO on 2 discrete stochastic environments (Nim and FrozenLake) but is clearly worse elsewhere, especially on continuous-control tasks (Swimmer: $\sim$110 vs. $\sim$260; InvertedPendulum: $\sim$194 vs. $\sim$756). Section 4.4 shows why naive multi-trajectory conditioning can hurt: when rollouts disagree, the Critic-LLM over-focuses on vivid worst-case traces, a phenomenon we call *salience bias*. R2PO's evidence design preserves the benefits of trajectory-grounded diagnosis while reducing this vulnerability.

4.4 Mechanistic analysis

The ablations raise a key question: *why* is trajectory-grounded revision so high-variance that it needs both selection and calibration? AlwaysCritic and CriticOnly perform similarly even though AlwaysCritic sees more information, and ThreeTraj, which sees *three* rollouts, does worse. To investigate, we focus on CartPole and Pong—where CriticOnly outperforms AlwaysCritic—and use AlwaysCritic as our main diagnostic, since it isolates trajectory-based revision without selection or calibration. For each environment, we analyze 1,000 revision decisions (10 runs, 100 iterations).

Trajectory-grounded diagnosis is high-variance. On Pong, 31.6% of AlwaysCritic iterations are regressions; on CartPole, 34%. The same diagnostic mechanism, the Critic-LLM reading a trajectory, identifying a failure point, and making a fix, produces both the largest improvements and regressions.

The root cause: unrepresentative trajectories. Under AlwaysCritic, the Critic-LLM sees only one rollout out of K=20. In stochastic environments, a single rollout is a *sample* of policy behavior, not the complete picture. When the trajectory reflects typical behavior, the Critic-LLM's diagnosis generalizes and the fix improves the policy. When the trajectory shows an atypical outcome (an unlucky termination, an uncharacteristic failure), the Critic-LLM diagnoses a non-existent problem and destroys behavior that was working in the 19 unseen rollouts. The Critic-LLM cannot distinguish these cases without information about how often the observed failure actually occurs.

Failure patterns. We categorize AlwaysCritic regressions into four patterns (full descriptions in Appendix I.1): *surgical fixes that backfire*, *compulsive revision of near-optimal policies*, *full redesigns on uninformative trajectories*, and *pure stochasticity*. The dominant pattern, compulsive revision, accounts for 12.6% of episodes on Pong and 14.2% on CartPole; pure stochasticity accounts for $< 1\%$, confirming noise is not the main driver.

Three trajectories and salience bias. We tested giving the Critic-LLM the best, worst, and median rollouts (the ThreeTraj variant) to address representativeness. This should, in principle, help distinguish systematic failures from outliers. On Pong, ThreeTraj achieves the worst mean delta per iteration ($+0.16$) despite the largest edits (mean edit distance 8.4), versus RepTraj's $+0.64$ at distance 6.2. On CartPole, the same pattern holds: ThreeTraj $+5.6$ at distance 4.6 vs. RepTraj $+16.0$ at distance 4.1.

To understand why, we coded all 2,000 ThreeTraj decisions on CartPole and Pong by scanning the Critic-LLM's reasoning text (Appendix I). We define a *salience-problem regression* as an episode where (i) the Critic-LLM's reasoning explicitly references the worst rollout, (ii) the worst rollout's return is strictly lower than the median (so the worst is genuinely an outlier), and (iii) the revision hurt performance ($\Delta < 0$). On CartPole, 233 of 304 regressions (76.6%) meet this strict definition. On Pong, where rollouts more often agree on a failure (worst $=$ median), ThreeTraj instead produces a positive mean delta of $+0.62$ on those episodes.

> **Salience bias.** *When shown multiple rollouts of a policy, the Critic-LLM systematically overweights the worst trajectory in its diagnosis and edits, even when most rollouts indicate good performance.*

We observe this directly. In a Pong episode, rollout rewards were 2.0, 3.0, 3.0 (mean 2.95). The Critic-LLM organized its revision around the worst rollout: "*From the worst rollout we saw that the policy never moved... this indicates that the do-nothing weights are too dominant.*" It changed six parameters; reward dropped to 0.15. Better-performing rollouts existed among the median and best, yet the Critic-LLM anchored on the worst trajectory. This pattern motivates R2PO's evidence design: failure frequency is conveyed through aggregate statistics rather than a worst-case trace.

R2PO produces focused, high-quality revisions. Table 5 reports revision outcomes across the three trajectory-evidence variants; full edit-pattern frequencies are in Appendix I.2. R2PO achieves the highest mean improvement per iteration ($+45.3$ on CartPole, $+0.87$ on Pong), substantially outperforming RepTraj ($+16.0$, $+0.64$) and ThreeTraj ($+5.6$, $+0.16$), while making the *smallest* edits (mean edit distance 1.6 on CartPole, 4.5 on Pong). It also achieves the highest acceptance rate (86–88%) and lowest regression rate (12–14%). The pattern breakdown attributes these gains: Near-Optimal Break drops $6\times$ on CartPole and $17\times$ on Pong (Revision rule); Surgical Backfire drops $7\times$ on CartPole and $3.5\times$ on Pong (rollout statistics preventing unrepresentative-trace fixes); productive Fine-Tune edits nearly triple on CartPole ($4.8\% \rightarrow 13.4\%$). Consistency across both environments confirms these mechanisms are not environment-specific.

The broader implication is that the main challenge in leveraging trajectories is not *whether* to use them, but *how* to summarize them so the LLM can distinguish systematic failures from rare salient mistakes.

Table 5: Revision summary statistics across 1,000 episodes per environment. **Best** per metric is highlighted. R2PO achieves the highest mean improvement while making the smallest edits.

	CartPole			Pong		
Metric	RepTraj	ThreeTraj	R2PO	RepTraj	ThreeTraj	R2PO
Mean Δ (reward/iter)	$+16.0$	$+5.6$	$\mathbf{+45.3}$	$+0.64$	$+0.16$	$\mathbf{+0.87}$
Mean edit distance	4.1	4.6	**1.6**	6.2	8.4	**4.5**
Revision accepted (%)	70.7	69.6	**86.3**	71.2	67.7	**87.6**
Regressions (%)	29.3	30.4	**13.7**	28.8	32.3	**12.4**

Qualitative examples. Figure 3 shows two representative R2PO episodes: a conservative one-parameter edit guided by statistics, and a well-reasoned minimal edit that destroys the policy and is rejected by selection. Two further examples (full policy rescues on Pong and FrozenLake) and full reasoning traces are in Appendix J.

5 Discussion and Conclusion

We introduced R2PO, a two-stage LLM framework that separates global policy proposal (Search-LLM) from local behavior-grounded revision (Critic-LLM), together with a calibrated trajectory

Example 1: Conservative Repair (CartPole, R2PO) 490.15 → 493.70 (+3.55)

Stats: Mean 490.15, Min 436, Max 500, Success 19/20

```
Initial:  [6.0, 5.5, 6.0, 6.0, -1.0, 6.0, -0.5, 6.0, -2.0, -2.0]
Revised: [6.0, 6.0, 6.0, 6.0, -1.0, 6.0, -0.5, 6.0, -2.0, -2.0]
```

Critic-LLM: *"The average reward (490.15) and success rate (19/20) indicate the policy is already highly effective... I increased only params[1]."*

Takeaway: Aggregate statistics guide a minimal one-parameter edit. ✓ **Accepted**

Example 2: Honest Failure (CartPole, R2PO) 436.05 → 173.30 (−262.75)

Stats: Mean 436.05, Success 15/20, sporadic failures

```
Initial:  [6.0, 6.0, 6.0, 6.0, -1.0, 6.0, -0.5, 6.0, -1.8, -2.0]
Revised: [6.0, 6.0, 6.0, 6.0, -1.0, 6.0, -0.3, 6.0, -1.5, -2.0]
```

Critic-LLM: *"The failures are sporadic; the median rollout is the same as with the current weights... The adjustments are small and target only the aspects linked to occasional failures."*

Takeaway: Selection catches a minimal edit that destroys the policy. × **Original kept**

Figure 3: Two representative R2PO revision episodes. Example 1 shows a conservative one-parameter edit guided by aggregate statistics; Example 2 shows why selection remains essential: well-reasoned minimal edits can still fail. Changed parameters are underlined. More examples in Appendix J.

evidence package and a selection rule. Our central finding is that rollout trajectories provide a fundamentally richer feedback channel for LLM-based policy optimization than scalar rewards alone, but only when presented carefully: naively passing trajectories to the reviser triggers *salience bias*, in which the LLM overweights vivid worst-case rollouts and over-corrects already-strong policies. R2PO's calibrated evidence package—median trajectory, aggregate statistics, and revision rule—mitigates this bias and yields a more stable and effective optimizer than scalar-only prompting-based search. Comparisons with classical black-box optimizers and evaluations on simulated robotic tasks further support R2PO's applicability to compact and structured policy classes. These results are achieved using a 20B open-weight model, indicating that the benefits of behavior-grounded reflective policy search are not limited to large proprietary systems.

Limitations. We focus on compact policy classes; extension to high-dimensional neural policies remains untested. R2PO also uses an environment-dependent threshold τ_c that currently requires coarse task-specific calibration. Although Appendix B shows robustness to $\pm 50\%$ variation and competitiveness without this rule, automatic calibration remains unresolved.

Future work. A first priority is extending R2PO beyond compact controllers. Promising directions include low-dimensional projections of neural network parameters (cf. [29], App. E) and structured policy classes such as compositional or hierarchical controllers, where parameter substructure could provide the Critic-LLM with semantically meaningful revision targets. A second direction is adaptive calibration of τ_c from rollout statistics and uncertainty signals, eliminating even the coarse manual setting required today. More broadly, salience bias may affect other settings in which LLMs reason over multiple evidence sources, such as multi-document QA, code review, and multi-turn agents; characterizing this bias beyond policy optimization remains open.

Acknowledgments and Disclosure of Funding

This work was supported by the Dubai Research, Development and Innovation (RDI) Grant Initiative, administered by the Dubai Future Foundation, under the CogWaters project (Project ID: 2025/DRDI0163). We thank Ignacio Carlucho, Radu-Casian Mihailescu, Adrian Turcanu, Kayvan Karim, and John Woodward for their valuable feedback and helpful discussions, and Aditya Shibu for developing the R2PO project website.

References

[1] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

[2] Onur Celik, Aleksandar Taranovic, and Gerhard Neumann. Acquiring diverse skills using curriculum reinforcement learning with mixture of experts. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 5907–5933. PMLR, 2024. URL https://proceedings.mlr.press/v235/celik24a.html.

[3] Aladin Djuhera, Swanand Ravindra Kadhe, Farhan Ahmed, Heiko Ludwig, and Holger Boche. TSR: Trajectory-Search Rollouts for Multi-Turn RL of LLM Agents, 2026. URL https://arxiv.org/abs/2602.11767.

[4] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods. *International Conference on Machine Learning*, pages 1587–1596, 2018.

[5] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *International Conference on Machine Learning*, pages 1861–1870, 2018.

[6] Yanqiao Han, Menglong Yang, Yang Ren, and Weizheng Li. Large Language Model Guided Reinforcement Learning Based Six-Degree-of-Freedom Flight Control. *IEEE Access*, 12: 89479–89492, 2024. doi: 10.1109/ACCESS.2024.3411015.

[7] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001. doi: 10.1162/106365601750190398.

[8] Nikolaus Hansen, Youhei Akimoto, and Petr Baudis. CMA-ES/pycma on github, 2019.

[9] Tim Head, Manoj Kumar, Holger Nahrstaedt, Gilles Louppe, and Iaroslav Shcherbatyi. scikit-optimize/scikit-optimize, 2021.

[10] Zhenyu Hou, Ziniu Hu, Yujiang Li, Rui Lu, Jie Tang, and Yuxiao Dong. TreeRL: LLM Reinforcement Learning with On-Policy Tree Search, 2025. URL https://arxiv.org/abs/2506.11902.

[11] Minae Kwon, Sang Michael Xie, Kalesha Bullard, and Dorsa Sadigh. Reward Design with Language Models, 2023. URL https://arxiv.org/abs/2303.00001.

[12] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *International Conference on Learning Representations*, 2016.

[13] Yecheng Jason Ma, William Liang, Hung-Ju Wang, Sam Wang, Yuke Zhu, Linxi Fan, Osbert Bastani, and Dinesh Jayaraman. DrEureka: Language Model Guided Sim-To-Real Transfer. In *Robotics: Science and Systems (RSS)*, 2024. URL https://arxiv.org/abs/2406.01967.

[14] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative Refinement with Self-Feedback. In *Advances in Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=S37hOerQLB.

[15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[16] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. *International Conference on Machine Learning*, pages 1928–1937, 2016.

[17] Ollama. gpt-oss:20b-cloud, 2025. URL https://ollama.com/library/gpt-oss:20b-cloud.

[18] OpenAI. Introducing gpt-oss, 2025. URL https://openai.com/index/introducing-gpt-oss/.

[19] Antonin Raffin. RL Baselines3 Zoo. https://github.com/DLR-RM/rl-baselines3-zoo, 2020.

[20] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[21] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust Region Policy Optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[22] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[23] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language Agents with Verbal Reinforcement Learning. In *Advances in Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=vAElhFcKW6.

[24] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, volume 25, 2012.

[25] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5033. IEEE, 2012.

[26] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A Standard Interface for Reinforcement Learning Environments. *arXiv preprint arXiv:2407.17032*, 2024.

[27] Xiefeng Wu. Enhancing Q-Learning with Large Language Model Heuristics, 2024. URL https://arxiv.org/abs/2405.03341.

[28] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large Language Models as Optimizers. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=Bb4VGOWELI.

[29] Yifan Zhou, Sachin Grover, Mohamed El Mistiri, Kamalesh Kalirathnam, Pratyush Kerhalkar, Swaroop Mishra, Neelesh Kumar, Sanket Gaurav, Oya Aran, and Heni Ben Amor. Prompted Policy Search: Reinforcement Learning through Linguistic and Numerical Reasoning in LLMs, 2025. URL https://arxiv.org/abs/2511.21928.

[30] Zihao Zhou, Bin Hu, Chenyang Zhao, Pu Zhang, and Bin Liu. Large Language Model as a Policy Teacher for Training Reinforcement Learning Agents. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence (IJCAI-24)*, pages 5671–5679, 2024. doi: 10.24963/ijcai.2024/627. URL https://arxiv.org/abs/2311.13373.

Appendix

Table of Contents

A Significance Tests for Mean-Reward Comparisons **14**

B Revision threshold sensitivity analysis **15**

C Stability analysis: best-vs-mean reward gap **17**

D Comparison with ProPS at extended training budget **17**

E R2PO algorithm **18**

F Revision threshold values **19**

G Full learning curves **19**

H Mean best reward for ablation variants **19**

I Salience analysis details **19**

I.1 Failure pattern definitions 23

I.2 Edit pattern frequencies 23

J Full qualitative reasoning traces **23**

K Custom environment specifications **27**

L R2PO Prompt Templates **28**

L.1 Fixed-value placeholders 28

L.2 Search-LLM Prompt (Continuous) 28

L.3 Search-LLM Prompt (Discrete) 29

L.4 Critic-LLM Prompt (Continuous) 29

L.5 Critic-LLM Prompt (Discrete) 30

M Dynamic ASV Debris Collection **31**

N Robotic Table Tennis **31**

A Significance Tests for Mean-Reward Comparisons

To complement the descriptive comparison of reward across methods, this appendix reports significance tests for both *mean reward* and *mean best reward*.

For each environment e and method m, let

$$x_{m,1}^{(e)}, x_{m,2}^{(e)}, \ldots, x_{m,n_m}^{(e)}$$

denote the *per-run* mean rewards, where each $x_{m,i}^{(e)}$ is the mean reward of run i for method m on environment e, averaged across all training iterations, and $n_m = 10$ in our experiments. Thus, the test compares the 10 run-level summary values for R2PO against the 10 run-level summary values for a baseline. The sample mean and (Bessel-corrected) sample variance for method m are

$$\bar{x}_m^{(e)} = \frac{1}{n_m} \sum_{i=1}^{n_m} x_{m,i}^{(e)}, \qquad \left(s_m^{(e)}\right)^2 = \frac{1}{n_m - 1} \sum_{i=1}^{n_m} \left(x_{m,i}^{(e)} - \bar{x}_m^{(e)}\right)^2.$$

The standard deviations reported in the main-paper descriptive tables (e.g., Tables 1 and 2) use this same Bessel-corrected sample-variance convention. The reported performance gap is

$$\Delta^{(e)} = \bar{x}_{\text{R2PO}}^{(e)} - \bar{x}_{\text{baseline}}^{(e)}.$$

Because the methods were evaluated with independent runs (rather than paired across methods) and exhibited unequal variances in several environments, we use two-sided Welch's t-tests.

$$t^{(e)} = \frac{\bar{x}_{\text{R2PO}}^{(e)} - \bar{x}_{\text{baseline}}^{(e)}}{\sqrt{\frac{\left(s_{\text{R2PO}}^{(e)}\right)^2}{n_{\text{R2PO}}} + \frac{\left(s_{\text{baseline}}^{(e)}\right)^2}{n_{\text{baseline}}}}},$$

with Welch–Satterthwaite degrees of freedom

$$\nu^{(e)} = \frac{\left(\frac{\left(s_{\text{R2PO}}^{(e)}\right)^2}{n_{\text{R2PO}}} + \frac{\left(s_{\text{baseline}}^{(e)}\right)^2}{n_{\text{baseline}}}\right)^2}{\frac{\left(\frac{\left(s_{\text{R2PO}}^{(e)}\right)^2}{n_{\text{R2PO}}}\right)^2}{n_{\text{R2PO}}-1} + \frac{\left(\frac{\left(s_{\text{baseline}}^{(e)}\right)^2}{n_{\text{baseline}}}\right)^2}{n_{\text{baseline}}-1}}.$$

The corresponding raw two-sided p-value is

$$p_{\text{raw}}^{(e)} = 2 \Pr\left(T_{\nu^{(e)}} \geq \left|t^{(e)}\right|\right).$$

Intuitively, a large positive t means that the R2PO runs, viewed as one group, have a higher mean than the baseline runs, viewed as another group, and that this gap is large relative to how much the runs fluctuate within each method. A negative t means the baseline group's mean is higher.

We compare R2PO against *all* core baselines for each environment. That is, for every environment we test R2PO against ProPS, ProPS+, and the corresponding SB3 baseline. Holm correction is then applied *separately within each baseline family across the 10 environments*; that is, one Holm correction for the 10 R2PO-vs-ProPS comparisons, one for the 10 R2PO-vs-ProPS+ comparisons, and one for the 10 R2PO-vs-SB3 comparisons. If the raw p-values within one such family, sorted in ascending order, are $p_{(1)}, \ldots, p_{(10)}$, the Holm-adjusted values are computed as

$$p_{(k)}^{\text{Holm}} = \max_{j \leq k} \left((10 - j + 1)\, p_{(j)}\right),$$

clipped to $[0, 1]$. The raw p-value answers whether a single comparison looks significant on its own; the Holm-adjusted p-value answers whether that comparison remains significant after accounting for the 10-environment family of comparisons against the same baseline type.

Table 6 reports the resulting comparisons for mean reward. The results are strong. After Holm correction, R2PO significantly outperforms 28 of the 30 core baseline comparisons. The only non-significant cases are MountainCarContinuous against SB3 (SAC), where SB3 remains slightly better

Table 6: Mean-reward significance tests comparing R2PO against all core baselines across the 10 environments. Each comparison uses a two-sided Welch's t-test over 10 independent runs per method; raw p-values are reported together with Holm-corrected p-values, with Holm correction applied separately within each baseline family across the 10 environments. Higher mean reward is highlighted. R2PO significantly outperforms 28 of the 30 core baseline comparisons after correction, with the only non-significant cases being MountainCarContinuous against SB3 (SAC), where SB3 is slightly better, and Maze against SB3 (A2C), where the methods are effectively tied in mean reward.

Environment	Baseline	Base mean	R2PO mean	Gap	Welch t	Raw p	Holm p
Nim	ProPS	-0.59	**0.61**	1.20	18.122	1.5e-08	1.1e-07
Nim	ProPS+	-0.25	**0.61**	0.86	34.411	1.5e-12	1.5e-11
Nim	SB3 (A2C)	0.01	**0.61**	0.60	5.958	0.0002	0.0008
Pong	ProPS	0.74	**2.51**	1.77	8.435	4.2e-06	1.7e-05
Pong	ProPS+	1.22	**2.51**	1.28	6.588	3.5e-05	0.0002
Pong	SB3 (PPO)	1.02	**2.51**	1.49	6.129	9.5e-05	0.0005
Swimmer	ProPS	89.22	**260.35**	171.13	8.934	9.6e-08	5.8e-07
Swimmer	ProPS+	162.05	**260.35**	98.29	4.130	0.0010	0.0031
Swimmer	SB3 (TRPO)	44.60	**260.35**	215.75	18.549	6.4e-09	5.1e-08
MountainCarCon	ProPS	-23.45	**81.61**	105.06	7.916	2.0e-05	6.0e-05
MountainCarCon	ProPS+	17.90	**81.61**	63.71	5.324	0.0004	0.0021
MountainCarCon	SB3 (SAC)	**82.33**	81.61	-0.72	-0.393	0.7004	1.0000
MountainCar	ProPS	-199.31	**-147.84**	51.47	21.891	3.2e-10	2.6e-09
MountainCar	ProPS+	-195.81	**-147.84**	47.97	18.897	3.7e-11	3.3e-10
MountainCar	SB3 (DQN)	-199.99	**-147.84**	52.15	23.196	2.4e-09	2.2e-08
InvDblPend	ProPS	79.44	**158.51**	79.07	3.553	0.0052	0.0052
InvDblPend	ProPS+	86.71	**158.51**	71.80	3.213	0.0091	0.0161
InvDblPend	SB3 (TRPO)	86.04	**158.51**	72.47	3.350	0.0085	0.0256
InvPendulum	ProPS	234.14	**756.08**	521.93	7.190	1.1e-06	5.7e-06
InvPendulum	ProPS+	309.52	**756.08**	446.55	4.834	0.0002	0.0013
InvPendulum	SB3 (TRPO)	24.35	**756.08**	731.73	14.977	1.1e-07	8.0e-07
FrozenLake	ProPS	0.05	**0.62**	0.57	20.858	6.5e-13	6.5e-12
FrozenLake	ProPS+	0.37	**0.62**	0.25	8.241	1.8e-07	1.4e-06
FrozenLake	SB3 (TRPO)	0.02	**0.62**	0.60	24.822	5.1e-11	5.1e-10
CartPole	ProPS	258.09	**474.67**	216.58	4.906	0.0008	0.0015
CartPole	ProPS+	253.06	**474.67**	221.61	5.214	0.0005	0.0021
CartPole	SB3 (TRPO)	216.92	**474.67**	257.75	12.434	1.6e-07	9.6e-07
Maze	ProPS	-1.03	**0.83**	1.86	31.357	4.2e-12	3.8e-11
Maze	ProPS+	0.76	**0.83**	0.07	3.001	0.0081	0.0161
Maze	SB3 (A2C)	0.83	**0.83**	0.00	0.016	0.9878	1.0000

in mean reward, and Maze against SB3 (A2C), where the two methods are effectively tied in mean reward. In contrast, R2PO significantly outperforms ProPS and ProPS+ on all 10 environments.

Table 7 reports the corresponding comparisons for mean best reward. Although only 10 of the 30 mean-best-reward comparisons remain significant after Holm correction, R2PO still matches or exceeds all core baselines descriptively, with positive gaps in most non-significant cases.

These tests reinforce, rather than replace, the descriptive reward tables. The main value of the significance analysis is to show that R2PO's higher mean reward is usually not an artifact of run-to-run noise.

B Revision threshold sensitivity analysis

A natural concern about the revision rule (Section 3.2) is whether R2PO's performance depends on careful per-environment tuning of the threshold τ_c. To address this, we conduct a sensitivity analysis on two environments, CartPole and Pong. For each environment, we run R2PO with three threshold settings, all matching the main results' 10-run protocol: (i) the default τ_c used in the main paper (set close to the known optimum, see Appendix F); (ii) $0.5 \cdot \tau_c$, halving the threshold to weaken the revision rule; and (iii) $\tau_c = -\infty$, which disables the rule entirely.

Table 7: Best-reward significance tests comparing R2PO against all core baselines across the 10 environments. Each comparison uses a two-sided Welch's t-test over 10 independent runs per method; raw p-values are reported together with Holm-corrected p-values, with Holm correction applied separately within each baseline family across the 10 environments. Higher mean best reward is highlighted. The best-reward differences are less uniformly significant than mean-reward differences: after correction, R2PO significantly outperforms 10 of the 30 core baseline comparisons, and still matches or exceeds all core baselines descriptively, with positive gaps in most non-significant cases.

Environment	Baseline	Base mean	R2PO mean	Gap	Welch t	Raw p	Holm p
Nim	ProPS	0.75	**1.00**	0.25	2.643	0.0268	0.1338
Nim	ProPS+	**1.00**	**1.00**	0.00	0.000	1.0000	1.0000
Nim	SB3 (A2C)	0.88	**1.00**	0.12	1.406	0.1934	0.5803
Pong	ProPS	2.15	**3.00**	0.85	2.833	0.0196	0.1177
Pong	ProPS+	2.78	**3.00**	0.22	1.687	0.1259	0.8815
Pong	SB3 (PPO)	2.33	**3.00**	0.67	2.681	0.0251	0.1257
Swimmer	ProPS	208.52	**294.57**	86.05	3.351	0.0043	0.0299
Swimmer	ProPS+	274.86	**294.57**	19.71	1.065	0.3011	1.0000
Swimmer	SB3 (TRPO)	67.89	**294.57**	226.68	14.771	1.8e-09	1.1e-08
MountainCarCon	ProPS	75.37	**98.75**	23.38	2.418	0.0387	0.1338
MountainCarCon	ProPS+	98.70	**98.75**	0.05	0.237	0.8155	1.0000
MountainCarCon	SB3 (SAC)	94.81	**98.75**	3.94	20.521	6.6e-14	5.2e-13
MountainCar	ProPS	-191.84	**-111.04**	80.80	9.790	3.1e-06	2.7e-05
MountainCar	ProPS+	-150.21	**-111.04**	39.17	4.603	0.0011	0.0115
MountainCar	SB3 (DQN)	-197.47	**-111.04**	86.44	50.957	7.6e-21	6.9e-20
InvDblPend	ProPS	112.18	**254.04**	141.86	1.923	0.0862	0.1724
InvDblPend	ProPS+	128.81	**254.04**	125.23	1.659	0.1281	0.8815
InvDblPend	SB3 (TRPO)	98.25	**254.04**	155.79	2.120	0.0630	0.2522
InvPendulum	ProPS	649.91	**1000.00**	350.09	2.562	0.0306	0.1338
InvPendulum	ProPS+	657.88	**1000.00**	342.12	2.434	0.0377	0.3394
InvPendulum	SB3 (TRPO)	28.49	**1000.00**	971.50	4298.503	1.0e-29	1.0e-28
FrozenLake	ProPS	0.24	**0.93**	0.69	14.707	1.2e-08	1.2e-07
FrozenLake	ProPS+	0.90	**0.93**	0.02	1.037	0.3146	1.0000
FrozenLake	SB3 (TRPO)	0.12	**0.93**	0.81	19.572	2.2e-10	1.5e-09
CartPole	ProPS	427.74	**500.00**	72.26	1.598	0.1446	0.1724
CartPole	ProPS+	396.21	**500.00**	103.79	2.115	0.0636	0.5086
CartPole	SB3 (TRPO)	490.76	**500.00**	9.24	1.000	0.3434	0.6869
Maze	ProPS	-0.70	**0.97**	1.66	6.000	0.0002	0.0016
Maze	ProPS+	**0.97**	**0.97**	0.00	0.000	1.0000	1.0000
Maze	SB3 (A2C)	**0.97**	**0.97**	0.00	0.000	1.0000	1.0000

Findings. R2PO's mean reward is robust across threshold settings, but the revision rule contributes meaningfully to training stability. On CartPole, halving τ_c reduces mean reward by $\sim$28 points and disabling it entirely costs $\sim$36 points; standard deviations grow correspondingly from 16.9 to 49.68 ($0.5\tau_c$) and 63.9 (disabled), reflecting more frequent destructive revisions. On Pong, mean reward is essentially preserved across settings, but the standard deviation roughly triples when the revision rule is disabled ($0.21 \rightarrow 0.59$), indicating that the rule's main effect on this environment is to suppress occasional regressions rather than to lift mean performance.

Notably, the disabled condition is not a naive baseline: it retains R2PO's two other calibration mechanisms (median-trajectory selection and aggregate rollout statistics). Even without the revision rule, R2PO substantially outperforms the strongest LLM-based baselines (Tables 1–2). This decomposition suggests that the median + statistics package provides the bulk of R2PO's gains by giving the Critic-LLM a representative view of policy behavior, while the rule adds a stability layer that prevents destructive revisions of already-strong policies.

These results suggest that practitioners can apply R2PO with a coarsely chosen τ_c (e.g., set near the known reward ceiling) without careful per-environment tuning. We discuss adaptive calibration as future work in Section 5.

Table 8: Revision threshold sensitivity. Each row reports mean and mean best reward across 10 independent runs for R2PO run with the listed τ_c value; *Default* matches the value used in the main paper. R2PO's mean reward degrades gracefully as τ_c is weakened or disabled, but variance increases substantially without the revision rule, particularly on CartPole.

Environment	Setting	τ_c	Mean reward	Mean best reward
CartPole	default	480	474.67 ± 16.90	500.00 ± 0.00
CartPole	$0.5\tau_c$	240	446.05 ± 49.68	489.84 ± 24.18
CartPole	disabled	$-\infty$	438.36 ± 63.90	492.49 ± 23.76
Pong	default	2.8	2.51 ± 0.21	3.00 ± 0.00
Pong	$0.5\tau_c$	1.4	2.51 ± 0.48	2.92 ± 0.24
Pong	disabled	$-\infty$	2.40 ± 0.59	2.94 ± 0.21

C Stability analysis: best-vs-mean reward gap

A method's gap between mean best reward and mean reward across training quantifies its training stability: a small gap indicates that strong policies, once discovered, are reliably maintained; a large gap indicates a method that occasionally finds good policies but loses them across iterations. We compute this gap as mean best reward − mean reward for each method-environment pair using the values from Tables 1 and 2 of the main paper.

Table 9: Stability gap (mean best reward − mean reward) per method across all 10 environments. Smaller gap indicates more stable training. **Smallest gap** per environment is highlighted. Note that small gaps may also reflect uniformly poor performance (e.g., a method that fails consistently has little to "lose" between best and mean); see discussion below.

Environment	ProPS	ProPS+	R2PO
Nim	1.34	1.25	**0.39**
Pong	1.41	1.56	**0.49**
Swimmer	119.30	112.81	**34.22**
MountainCarCon	98.82	80.80	**17.14**
MountainCar	**7.47***	45.61	36.80
InvDblPend	**32.75***	42.10	95.53
InvPendulum	415.77	348.35	**243.92**
FrozenLake	**0.19***	0.54	0.31
CartPole	169.65	143.15	**25.33**
Maze	0.34	0.21	**0.14**

* Cases where the smallest stability gap reflects uniformly poor performance rather than reliable stability: ProPS reaches near-floor performance on these environments (e.g., MountainCar mean reward ~ -199 very close to its best ~ -192 , indicating that the method never finds good policies, rather than stably maintaining them).

R2PO exhibits the smallest stability gap on 7 of 10 environments. The three exceptions (MountainCar, InvDblPend, FrozenLake) are not cases where ProPS is genuinely more stable, but rather cases where ProPS reaches near-floor performance and therefore has little spread between its best and mean rewards. On environments where ProPS finds good policies at all (Pong, Swimmer, MountainCarCon, InvPendulum, CartPole), R2PO's gap is consistently smaller. This confirms the qualitative claim in Section 4.2 that R2PO reliably sustains strong policies once discovered, while prior LLM-based methods exhibit substantially higher run-to-run variability.

D Comparison with ProPS at extended training budget

To contextualize R2PO against the originally published ProPS results, Table 10 reports our mean best reward (4,000 episodes per run, gpt-oss:20b) alongside the ProPS, ProPS+, and SB3 baseline numbers from [29] (8,000 episodes per run; GPT-4o for ProPS and ProPS+, 10 independent runs).

We use the same Gymnasium versions, custom environment specifications, and mean-best-reward metric as [29]. The ProPS and ProPS+ numbers in this table differ from the corresponding entries in Tables 1 and 2 of the main paper, which report our independent matched-budget reproductions (4,000 episodes, gpt-oss:20b, 10 independent runs).

Despite training on half the data and using an open-weight 20B-parameter model rather than GPT-4o, R2PO matches or exceeds both prior LLM-based methods and the strongest SB3 baseline on 9 of 10 environments. The exception is Inverted Double Pendulum, where TRPO's reported mean of 3609 has a standard error of 4000, indicating one or two outlier runs rather than reliable performance. This result demonstrates that the gains from trajectory-grounded revision are not contingent on access to large proprietary models or extended training budgets.

Table 10: Mean best reward ± standard error of the mean over 10 independent runs, matching the convention of [29]. Best Baseline, 2nd Best Baseline, ProPS, and ProPS+ numbers are reproduced from [29] (8,000 episodes per run; GPT-4o for ProPS and ProPS+, 10 runs). R2PO numbers are from our experiments (4,000 episodes per run, gpt-oss:20b, 10 runs). **Best** per environment is highlighted across all reported methods.

Domain	Best Baseline	2nd Best Baseline	ProPS	ProPS+	R2PO
Nim	A2C 0.58 ± 0.10	TRPO 0.50 ± 0.10	0.33 ± 0.29	0.97 ± 0.09	$\mathbf{1.00 \pm 0.00}$
Pong	PPO 2.29 ± 0.91	TRPO 1.36 ± 1.05	2.80 ± 0.26	2.99 ± 0.03	$\mathbf{3.00 \pm 0.00}$
Swimmer	TRPO 52.96 ± 18.86	A2C 39.40 ± 6.54	218.83 ± 58.45	227.30 ± 56.23	$\mathbf{294.57 \pm 13.85}$
MountainCarCon	SAC 86.65 ± 0.84	PPO 78.16 ± 5.32	87.21 ± 29.28	89.16 ± 29.72	$\mathbf{98.75 \pm 0.14}$
MountainCar	DQN -194.36 ± 1.47	A2C -200.00 ± 0.00	-126.11 ± 21.67	-116.71 ± 15.20	$\mathbf{-111.04 \pm 1.24}$
InvDblPend	TRPO $\mathbf{3609.37 \pm 4000.04}$	PPO 108.60 ± 4.12	128.17 ± 24.52	148.39 ± 48.65	254.04 ± 73.49
InvPendulum	TRPO 571.31 ± 358.88	PPO 218.65 ± 129.31	$\mathbf{1000.00 \pm 0.00}$	$\mathbf{1000.00 \pm 0.00}$	$\mathbf{1000.00 \pm 0.00}$
FrozenLake	TRPO 0.22 ± 0.05	PPO 0.16 ± 0.02	0.57 ± 0.17	0.19 ± 0.05	$\mathbf{0.93 \pm 0.02}$
CartPole	TRPO 465.34 ± 62.32	PPO 365.86 ± 73.38	478.27 ± 65.17	$\mathbf{500.00 \pm 0.00}$	$\mathbf{500.00 \pm 0.00}$
Maze	A2C $\mathbf{0.97 \pm 0.00}$	TRPO $\mathbf{0.97 \pm 0.00}$	0.55 ± 0.83	$\mathbf{0.97 \pm 0.00}$	$\mathbf{0.97 \pm 0.00}$

E R2PO algorithm

Algorithm 1 provides the full R2PO optimization loop. At each iteration, the Search-LLM proposes an initial parameter vector from the reward-only replay history, the environment evaluates it over K rollouts, and the resulting trajectories are summarized into a calibrated evidence package (median trajectory, aggregate statistics, and a revision rule controlled by τ_c). The Critic-LLM uses this evidence to propose a revised parameter vector, which is re-evaluated; the keep-best selection rule retains whichever candidate achieves the higher mean reward and appends it to the replay buffer.

Algorithm 1 R2PO

Require: environment ENV, environment description `env_desc`, rollouts per evaluation K, iterations T, Search-LLM prompt P_1, Critic-LLM prompt P_2, revision threshold τ_c

REPLAY_BUFFER $\leftarrow \emptyset$
for $t = 1, \ldots, T$ **do**
  $H \leftarrow$ FORMATHISTORY(REPLAY_BUFFER) ▷ reward-only: $\{(\theta_i, \bar{R}_i)\}$
  $\theta_{\text{init}} \leftarrow$ SEARCH-LLM(P_1, H) ▷ Stage 1: proposal
  $\bar{R}_{\text{init}}, \mathcal{T}_{\text{init}} \leftarrow$ EVAL(ENV, θ_{init}, K)
  $\tau^* \leftarrow$ SELECTMEDIAN$(\mathcal{T}_{\text{init}})$
  stats $\leftarrow$ AGGREGATESTATS$(\mathcal{T}_{\text{init}})$
  evidence $\leftarrow$ BUILDCONTEXT$(\theta_{\text{init}}, \bar{R}_{\text{init}}, \tau^*, \text{stats}, H, \tau_c, \texttt{env_desc})$
  $\theta_{\text{rev}} \leftarrow$ CRITIC-LLM$(P_2, \text{evidence})$ ▷ Stage 2: reflection
  $\bar{R}_{\text{rev}} \leftarrow$ EVAL(ENV, θ_{rev}, K)
  if $\bar{R}_{\text{rev}} \geq \bar{R}_{\text{init}}$ **then** ▷ selection
    REPLAY_BUFFER.ADD$(\theta_{\text{rev}}, \bar{R}_{\text{rev}})$
  else
    REPLAY_BUFFER.ADD$(\theta_{\text{init}}, \bar{R}_{\text{init}})$
  end if
end for
return REPLAY_BUFFER

F Revision threshold values

Table 11 reports the revision threshold τ_c used by R2PO for each environment. The threshold is set close to the strongest reward empirically achievable in our setup, which equals the theoretical optimum on tasks with bounded returns and is set lower on tasks where the formal ceiling is far above what any method reaches under our budget.

Table 11: Revision threshold τ_c for each environment. Set close to the strongest reward empirically achievable in our setup; equals the theoretical optimum on bounded-return tasks (Nim, Pong, MountainCar, FrozenLake, CartPole, Maze) and falls below it on tasks with very high theoretical ceilings (e.g., InvDblPend, where $\tau_c = 450$ vs. a formal optimum of $10{,}000$).

Environment	τ_c	Optimal reward
Nim	0.95	1.00*
Pong	2.80	3.00*
Swimmer	230	250*
MountainCarCon	97	100§
MountainCar	−120	−120*
InvDblPend	450	10000§
InvPendulum	950	1000§
FrozenLake	0.85	1.00‡
CartPole	480	500†
Maze	0.90	0.97*

† official Gymnasium reward threshold (and, for CartPole-v1, also the effective episode maximum); ‡ episode maximum return; § theoretical reward ceiling implied by the reward function and horizon, not an official Gymnasium threshold; * paper reference reward used to set τ_c; official Gymnasium thresholds, where available, may differ.

G Full learning curves

Figure 4 shows learning curves (mean $\pm$ standard deviation over 10 independent runs) for R2PO, ProPS, ProPS+, and the best SB3 baseline across all ten environments. R2PO generally reaches strong performance earlier and maintains it more consistently than ProPS and ProPS+, which often exhibit higher variance and less stable training.

H Mean best reward for ablation variants

Table 12 reports mean best reward across 10 independent runs for R2PO and all ablation variants. The pattern matches mean reward (Table 4 in the main text): R2PO matches or exceeds the strongest variant on all environments.

I Salience analysis details

To quantify the salience bias observed in the ThreeTraj variant, we systematically analyzed all 2,000 ThreeTraj episodes across CartPole and Pong (1,000 each, 10 independent runs per environment). For each episode, we parsed Critic-LLM's reasoning text from the logged output, checking whether the worst rollout was explicitly mentioned, and computed the reward delta $\Delta = R_{\text{revised}} - R_{\text{initial}}$.

Operational definition. An episode is classified as a *salience-problem regression* if all of the following hold:

1. Critic-LLM's reasoning explicitly references the worst rollout.
2. The worst rollout's return is strictly lower than the median (*strict* criterion), so the worst is genuinely an outlier rather than the modal outcome.
3. The revision hurt performance: $\Delta < 0$.

We also report a *permissive* variant that relaxes condition (2) to require only that the worst be no greater than the median and strictly less than the best. The strict criterion isolates outlier-driven

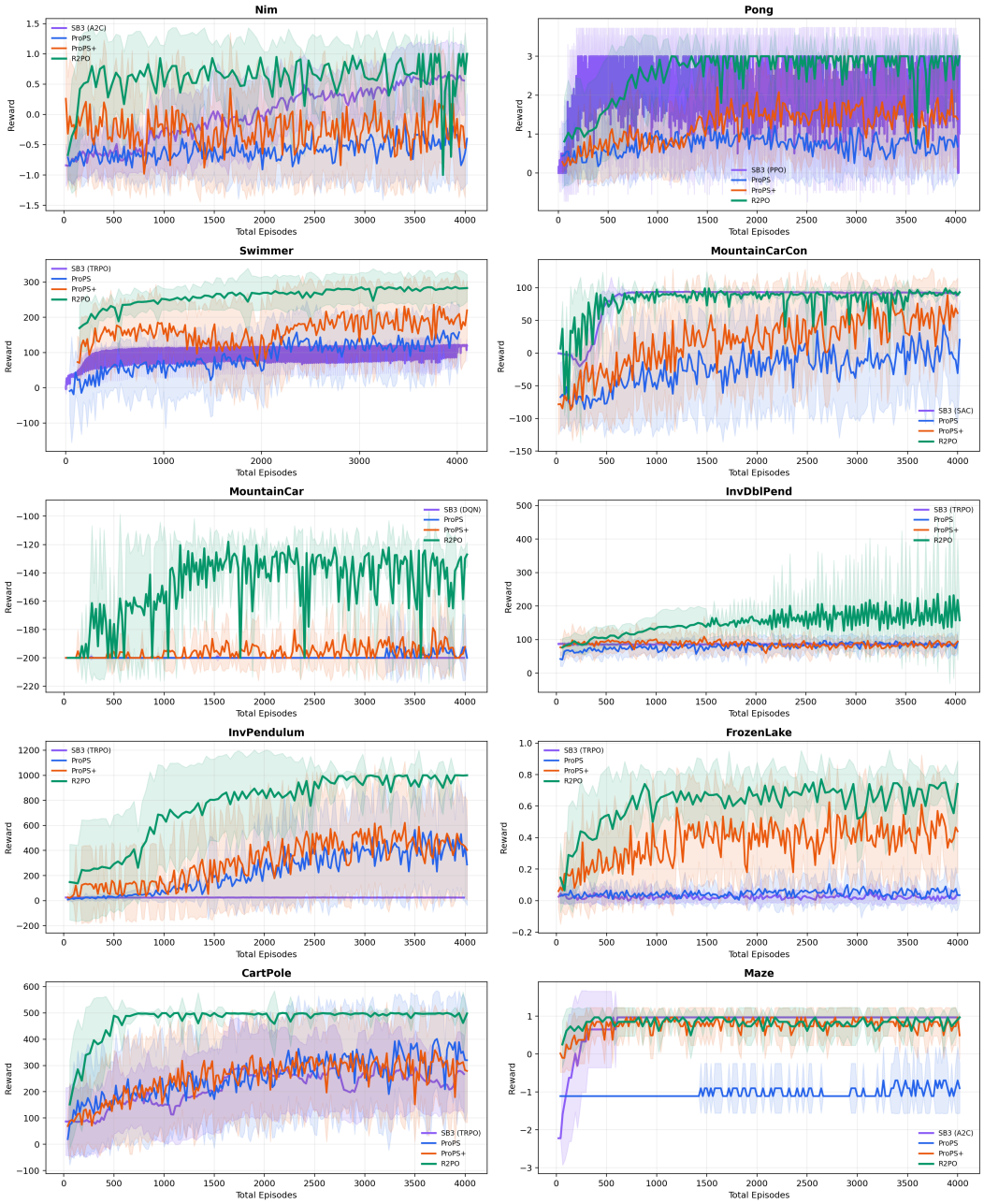


Figure 4: Learning curves (mean $\pm$ standard deviation over 10 independent runs) for R2PO, ProPS, ProPS+, and the best SB3 baseline across all ten environments.

misdiagnosis; the permissive criterion additionally counts cases where the worst trajectory reflects a $\geq 50\%$ failure mode but is still cited by the Critic-LLM as the central evidence for revision.

Results. The strict salience proxy yields different rates on CartPole (76.6%) and Pong (31.3%) because the two environments differ in how often the worst rollout is genuinely an outlier. On CartPole, all 304 regression episodes have worst $<$ median, so the strict and permissive criteria coincide: in 76.6% of regression cases, the Critic-LLM cited an outlier worst-case rollout and produced a harmful revision. On Pong, 61.5% of regression episodes have worst $=$ median, where the worst trajectory is also typical and Critic-LLM's reference to it is closer to legitimate diagnosis; the strict criterion correctly excludes these cases, leaving 31.3% as true outlier-driven regressions.

Table 12: Mean best reward ($\pm$ standard deviation) across 10 independent runs for R2PO and the ablation variants, where each run's best is taken across training iterations. **Best** per environment is highlighted. BestBaseline denotes the best among ProPS, ProPS+, and all SB3 algorithms. R2PO matches or exceeds the strongest variant on all environments.

Environment	BestBaseline	PureSearch	ActorSecondPass	CriticOnly	AlwaysCritic	RepTraj	ThreeTraj	R2PO
Nim	$\mathbf{1.00 \pm 0.00}$ (ProPS+)	0.70 ± 0.29	0.45 ± 0.46	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$
Pong	2.78 ± 0.41 (ProPS+)	2.31 ± 0.62	2.18 ± 0.63	$\mathbf{3.00 \pm 0.00}$	2.83 ± 0.54	3.00 ± 0.02	2.74 ± 0.41	$\mathbf{3.00 \pm 0.00}$
Swimmer	274.86 ± 38.81 (ProPS+)	219.56 ± 59.49	175.38 ± 83.16	210.37 ± 77.23	264.22 ± 72.99	257.80 ± 63.12	178.71 ± 92.87	$\mathbf{294.57 \pm 43.79}$
MountainCarCon	98.70 ± 0.50 (ProPS+)	95.31 ± 7.96	75.40 ± 40.01	83.52 ± 29.86	70.55 ± 42.99	46.70 ± 49.24	68.90 ± 47.57	$\mathbf{98.75 \pm 0.44}$
MountainCar	-150.21 ± 26.62 (ProPS+)	-170.22 ± 41.59	-191.71 ± 26.22	-114.68 ± 4.04	-117.16 ± 7.37	-128.69 ± 32.99	-118.60 ± 16.68	$\mathbf{-111.04 \pm 3.92}$
InvDblPend	128.81 ± 54.51 (ProPS+)	121.84 ± 39.64	112.29 ± 16.56	232.67 ± 106.26	207.42 ± 86.19	211.30 ± 91.05	124.09 ± 38.11	$\mathbf{254.04 \pm 232.39}$
InvPendulum	657.88 ± 444.42 (ProPS+)	720.46 ± 451.28	810.05 ± 400.62	365.45 ± 380.21	287.69 ± 273.42	815.09 ± 298.26	357.20 ± 445.93	$\mathbf{1000.00 \pm 0.00}$
FrozenLake	0.90 ± 0.04 (ProPS+)	0.29 ± 0.17	0.24 ± 0.08	0.90 ± 0.04	0.88 ± 0.06	0.91 ± 0.03	0.91 ± 0.04	$\mathbf{0.93 \pm 0.05}$
CartPole	490.76 ± 29.22 (TRPO)	440.09 ± 129.94	461.59 ± 99.44	498.62 ± 3.45	498.54 ± 4.62	487.74 ± 38.79	$\mathbf{500.00 \pm 0.00}$	$\mathbf{500.00 \pm 0.00}$
Maze	$\mathbf{0.97 \pm 0.00}$ (ProPS+)	-0.90 ± 0.66	-0.90 ± 0.66	$\mathbf{0.97 \pm 0.00}$	$\mathbf{0.97 \pm 0.00}$	$\mathbf{0.97 \pm 0.00}$	$\mathbf{0.97 \pm 0.00}$	$\mathbf{0.97 \pm 0.00}$

Table 13: Salience analysis across 2,000 ThreeTraj episodes (1,000 each on CartPole and Pong, 10 runs per environment). Under the strict criterion, salience-problem regressions account for 76.6% of CartPole regressions and 31.3% of Pong regressions. The gap reflects task structure: on CartPole, all 304 regressions have worst $<$ median, so strict and permissive criteria coincide; on Pong, 61.5% of regressions have worst $=$ median, where the worst trajectory is a typical failure mode rather than an outlier.

Metric	CartPole	Pong
Total episodes	1,000	1,000
Total regressions	304 (30.4%)	323 (32.3%)
Salience-problem regressions (% of regressions)		
Strict (worst $<$ median)	233 (76.6%)	101 (31.3%)
Permissive (worst $\leq$ median $<$ best)	233 (76.6%)	261 (80.8%)

The Pong rollout-agreement breakdown (Table 14) further supports the diagnosis: when failures are confirmed across rollouts (worst $=$ median), ThreeTraj produces a positive mean delta ($+0.62$), indicating that trajectory-grounded diagnosis works well when the worst trajectory is representative. When rollouts suggest the policy is already strong (preservation-favored), the mean delta is -0.35, indicating that Critic-LLM tends to damage policies it should have preserved. Together, these results support the design decision in R2PO to exclude the worst-case trajectory from Critic-LLM's context and instead provide failure frequency through aggregate rollout statistics.

Table 14: Pong rollout-agreement breakdown of the 1,000 ThreeTraj episodes. The mean revision delta varies sharply with whether the worst rollout is representative of typical policy behavior.

Rollout-agreement structure	Episodes	Mean Δ
Failure-confirmed (worst $=$ median $<$ best)	491	$+0.62$
Preservation-favored (median $=$ best $>$ worst)	382	-0.35
All-diverse (worst $<$ median $<$ best)	127	-0.09

Qualitative examples of salience-driven regressions. We provide three representative ThreeTraj episodes where Critic-LLM anchors on the worst rollout despite stronger evidence from the median and/or best rollouts. In all three cases, the initial policy is already strong on at least two of the three rollouts, Critic-LLM diagnoses the failure primarily through the worst rollout, and the revision reduces the mean reward. Traces are copied verbatim from logged output files, with key evidence phrases highlighted.

Salience Example 1: Full Reasoning (Pong, run 10, episode 92)

Rollouts: worst $= 2.0$, median $= 3.0$, best $= 3.0$ **Initial:** 2.95 **Revised:** 0.15 $\Delta = -2.80$

Critic-LLM output:

```
params[0]:  2.4, params[1]:  -3.0, params[2]:  0.0, params[3]:  1.0, params[4]:
-1.0, params[5]:  0.4, params[6]:  -1.5, params[7]:  1.0, params[8]:  0.0, params[9]:
```

0.8, params[10]: 0.0, params[11]: 0.8, params[12]: 0.0, params[13]: 0.8, params[14]: 0.0, params[15]: -1.0, params[16]: 1.0, params[17]: -1.0

> I only adjusted the weights that directly influence when the paddle chooses to move up or down versus staying still. From the worst rollout we saw that the policy never moved (action 2 every step), even as the ball's y-coordinate changed steadily. This indicates that the "do-nothing" weights are too dominant, especially the positive weight on ball x (param 5 = 0.8) and the negative weight on ball vx (param 11 = -0.8). In the median and best rollouts the paddle sometimes moves at the end of the episode, but the long-term reward only reaches 3.0, so a modest shift toward proactive motion should raise the average reward without over-reacting.
>
> Specific changes: param 5 reduced from 0.8 to 0.4; param 6 increased from -2.8 to -1.5; param 7 reduced from 2.8 to 1.0; param 10 set to 0.0; param 11 changed sign from -0.8 to $+0.8$; param 17 moved from 0.0 to -1.0.

Salience evidence: Critic-LLM organized all six parameter changes around the worst rollout's "never moved" behavior, despite the median and best rollouts already achieving the maximum reward of 3.0. The result was a catastrophic drop from 2.95 to 0.15.

Salience Example 2: Full Reasoning (Pong, run 8, episode 9)

Rollouts: worst $= 0.0$, median $= 3.0$, best $= 3.0$ **Initial:** 2.75 **Revised:** 0.15 $\Delta = -2.60$

Critic-LLM output:

params[0]: 1.5, params[1]: -1.5, params[2]: -0.5, params[3]: 0.5, params[4]: -0.5, params[5]: 0.5, params[6]: -1.0, params[7]: 1.0, params[8]: -1.0, params[9]: 1.0, params[10]: -1.0, params[11]: 1.0, params[12]: -0.5, params[13]: 0.5, params[14]: -0.5, params[15]: 0.5, params[16]: -0.5, params[17]: 0.5

> I kept the bias terms moderate to prevent the action scores from drifting too high or low. The weight pattern now favors the paddle to move up when the ball's y-coordinate is below the paddle's y, and to move down when it is above. This mirrors the behavior seen in the successful median/best rollouts.
>
> In the worst rollout, the ball was moving upward (vy = 3.31) but the policy's current weights caused it to over-react to the ball's x and y independently, making the paddle lag behind the ball's trajectory. The new weight set should reduce that lag.
>
> The training trial with parameters near 2.0 (trial 26) achieved a high mean_reward of 2.90; the new parameters preserve that general structure while slightly amplifying the paddle_y influence. This focused adjustment should bring the average reward closer to the target of $\sim$3 while avoiding over-fitting to the rare failure case seen only in rollout 17.

Salience evidence: Critic-LLM acknowledges the failure was "rare" and "seen only in rollout 17," yet restructures the entire weight pattern to address it. Two rollouts already achieved the maximum reward (3.0), but the revision was driven by the single zero-reward rollout.

Salience Example 3: Full Reasoning (CartPole, run 5, episode 67)

Rollouts: worst $= 446$, median $= 500$, best $= 500$ **Initial:** 497.30 **Revised:** 9.55 $\Delta = -487.75$

Critic-LLM output:

```
params[0]: 1.5, params[1]: -0.6, params[2]: -4.8, params[3]: 1.0, params[4]:
-1.1, params[5]: -1.2, params[6]: 2.8, params[7]: 2.5, params[8]: 2.6, params[9]:
2.4
```

> The worst rollout repeatedly toggled actions when the pole angle was only a few degrees off-balance, leading to a reward drop to ∼446. In the best and median trajectories the agent consistently chose the action that kept the pole upright even for small angle deviations.
>
> I reduced the magnitude of both pole-angle weights: params[4] from −1.3 to −1.1, and params[5] from −1.4 to −1.2. This makes the decision boundary for the angle less steep, allowing the policy to keep the same action longer when the pole is close to upright.
>
> Additionally the worst rollout tended to "over-react" after large cart velocity spikes. Lowering the bias of the right-push column (params[9]) from 2.6 to 2.4 forces the agent to be slightly more cautious.
>
> A slightly higher weight on the cart-position component for the left-push column (params[0] to 1.5) and a modest increase in the cart-angle weight for the right-push column (params[7] to 2.5) help balance the left and right pushes, yielding smoother trajectories.

Salience evidence: Critic-LLM changed five parameters, all motivated by the worst rollout's behavior ("toggled actions," "over-react"). The policy was nearly perfect (mean 497.3, median and best both 500). The result was the most catastrophic regression in our analysis: from 497.30 to 9.55.

I.1 Failure pattern definitions

We categorize AlwaysCritic regressions into four patterns:

- *Surgical fixes that backfire*: a 1–3 parameter edit is locally correct for the observed trace but disrupts behavior in other rollouts.
- *Compulsive revision of near-optimal policies*: the Critic-LLM perturbs a high-reward policy citing risks that have not manifested (12.6% of episodes on Pong, 14.2% on CartPole).
- *Full redesigns on uninformative trajectories*: near-zero policies whose short rollout traces prompt the Critic-LLM to redesign the policy from scratch.
- *Pure stochasticity*: identical revisions yield different rewards due to rollout variance ($< 1\%$ of episodes).

I.2 Edit pattern frequencies

Table 15 reports the frequency of distinct edit patterns across the three trajectory-evidence variants. Regression patterns (Near-Optimal Break, Surgical Backfire) capture cases where the Critic-LLM's revision hurts performance; improvement patterns (Pinpoint Fix, Fine-Tune) capture cases where it helps. Pattern definitions are in Appendix I.1.

J Full qualitative reasoning traces

We provide complete Critic-LLM reasoning output for the two examples in Figure 3, plus three additional revision episodes: two full policy rescues (Pong: $0.05 \rightarrow 3.00$ and FrozenLake: $0.00 \rightarrow 0.90$) and a loop-diagnosis case. Traces are copied verbatim from logged output files, with key evidence phrases highlighted.

Conservative Repair (CartPole, run 10, episode 26)

Initial reward: 490.15 **Revised reward:** 493.70 $\Delta = +3.55$

Stats given to Critic-LLM:

Table 15: Edit pattern frequencies (% of 1,000 episodes). **Best** per pattern is highlighted. R2PO sharply reduces destructive patterns while increasing productive ones on both environments.

	CartPole			Pong		
Pattern	RepTraj	ThreeTraj	R2PO	RepTraj	ThreeTraj	R2PO
Regression patterns (lower is better)						
Near-Optimal Break	13.3	18.6	**2.1**	10.5	6.0	**0.6**
Surgical Backfire	6.0	1.7	**0.8**	2.8	1.2	**0.8**
Improvement patterns (higher is better)						
Pinpoint Fix	4.1	**6.8**	6.2	3.5	1.6	**3.7**
Fine-Tune	4.8	8.1	**13.4**	0.1	0.0	**1.3**

```
Reward:  mean=490.15, min=303.00, max=500.00
Episode length:  mean=490.1, min=303, max=500
Success rate:  19/20 rollouts reached reward=500.00
Failure rate:  1/20 rollouts finished below reward=500.00
Median rollout (rollout 0, reward=500.0000, length=500)
```

Critic-LLM reasoning:

```
params[0]:  6, params[1]:  6, params[2]:  6, params[3]:  6,
params[4]:  -1, params[5]:  6, params[6]:  -0.5, params[7]:  6,
params[8]:  -2, params[9]:  -2
The one-step failure at 303 was caused by a slightly
weaker push to the right (action 1) when the pole angle
was slightly positive and the cart velocity was positive.
In the median trajectory this situation never caused an
issue; the policy performed robustly for all 500 steps.
The average reward (490.15) and success rate (19/20)
indicate that the policy is already highly effective, with only
one occasional sub-maximal episode.
To eliminate this outlier without risk of over-fitting, I increased only
params[1], which controls the contribution of cart velocity
to the left-push action, from 5.5 to the maximum of 6.0.  This
mirrors the successful configuration from trial 34, which
achieved perfect 500-step episodes across all rollouts.
All other parameters are kept identical to preserve the observed strong performance.
```

Full Rescue (Pong, run 10, episode 86)

Initial reward: 0.05 **Revised reward:** 3.00 $\Delta = +2.95$

Stats given to Critic-LLM:

```
Reward:  mean=0.05, min=0.00, max=1.00
Episode length:  mean=180.0, min=81, max=541
Success rate:  1/20 rollouts reached reward=1.00
Failure rate:  19/20 rollouts finished below reward=1.00
Median rollout (rollout 0, reward=0.0000, length=233)
```

Critic-LLM reasoning:

```
params[0]:  0.9, params[1]:  -0.9, params[2]:  0.0, params[3]:  0.0,
params[4]:  0.0, params[5]:  0.0, params[6]:  -0.9, params[7]:  0.9,
params[8]:  0.0, params[9]:  0.0, params[10]:  0.0, params[11]:  0.0,
params[12]:  0.0, params[13]:  0.0, params[14]:  0.0, params[15]:
0.0, params[16]:  0.0, params[17]:  0.0
The current policy almost never moves the paddle because
the weight on ball_x (params[3]) pulls action 0 down
```

as the ball travels rightward, while the action-2 "do nothing" score is always higher than the up/down scores. The median rollout shows the paddle staying at y = 300 for most of the episode even though the ball's y grows steadily from 300 to ~500, so the relative vertical distance is never used to drive a move. The aggregate statistics (many successful trials) indicate that the best performing policies depend only on the paddle's y position and the ball's y position, with the following relation:

Action 0 (up) score = pad_y - ball_y
Action 1 (down) score = -pad_y + ball_y
Action 2 (stay) score = 0

This simple comparison perfectly tracks the ball's vertical motion and yields a reward of +1 in every episode.

The new proposal keeps this structure but chooses a scaling that has not been tried before (0.9 instead of the exact 1.0 or 1.2 used in earlier trials). Because both action 0 and action 1 are scaled equally, the relative ordering of their scores is unchanged, ensuring identical behaviour while satisfying the rule against reproducing existing parameter sets.

Full Rescue (FrozenLake, run 10, episode 10)

Initial reward: 0.00 **Revised reward:** 0.90 $\Delta = +0.90$

Stats given to Critic-LLM:

Reward: mean=0.00, min=0.00, max=0.00
Episode length: mean=100.0, min=100, max=100
Success rate: 0/20 rollouts reached reward=1.00
Failure rate: 20/20 rollouts finished below reward=1.00
Median rollout (rollout 0, reward=0.0000, length=100, outcome=reached the rollout cap)

Critic-LLM reasoning:

params[0]: 0, params[1]: 3, params[2]: 0, params[3]: 3,
params[4]: 0, params[5]: 0, params[6]: 0, params[7]: 1,
params[8]: 3, params[9]: 1, params[10]: 0, params[11]: 2,
params[12]: 0, params[13]: 2, params[14]: 1, params[15]: 3

I revisited the "median rollout" and the average statistics to pinpoint where the all-3 policy failed. * The agent kept attempting the "left/up/right" mix (action 3), causing it to repeatedly bounce in place or steer into holes with probability 1/3. * States near the bottom and right edge (7, 11, 14) were exposed to a high chance of stepping into a hole when using action 0 or 3. I therefore kept the successful choices from Trial 2 (e.g., 0 at state 0, 3 at state 1, etc.) but made two targeted changes that reduce hole-risk:

1. State 7 - originally action 3 had a 1/3 chance of moving up into hole 3. Replaced by action 1, which only risks moving left to safe state 6 or staying, eliminating the up-hole accidental step.
2. State 11 - remaining in column 3, action 0 risked falling left into hole 10. Switched to action 2, which keeps the agent within the goal corridor (up to 7 or down to 15) and never reaches the hole directly.

All other actions retain the values that previously yielded high reward (0.8 in Trial 2). This modest, focused adjustment is expected to increase the success rate toward the target ≈ 1.0 while

```
avoiding the stochastic pitfalls that caused the all-3 policy to
fail.
```

Honest Failure (CartPole, run 10, episode 16)

Initial reward: 436.05 **Revised reward:** 173.30 $\Delta = -262.75$

Stats given to Critic-LLM:

```
Reward:  mean=436.05, min=204.00, max=500.00
Episode length:  mean=436.1, min=204, max=500
Success rate:  15/20 rollouts reached reward=500.00
Failure rate:  5/20 rollouts finished below reward=500.00
Median rollout (rollout 0, reward=500.0000, length=500)
```

Critic-LLM reasoning:

```
params[0]:  6, params[1]:  6, params[2]:  6, params[3]:  6,
params[4]:  -1, params[5]:  6, params[6]:  -0.3, params[7]:  6,
params[8]:  -1.5, params[9]:  -2
The collected statistics show a mean reward of 436.05 with a
75% success rate (15/20 rollouts reached the full 500 step).
The failures are sporadic;  the median rollout (full length 500)
is the same as with the current weights , indicating that the policy
is essentially correct but occasionally pushes the cart too hard at
the start of some episodes.
Inspecting the median trajectory, the cart velocity term (params[1]
and params[3]) dominates the action decision together with a large
bias difference between the two actions (params[8] and params[9]).
When the initial cart velocity is slightly high, the action chosen
can generate a large impulse that drives the cart beyond the allowed
bounds early in the episode.
To temper this over-reaction we reduce the magnitude of the
theta-velocity weight in the first action (params[6]).  A smaller
negative value (-0.3 instead of -0.5) keeps the swing behavior
largely the same while slightly decreasing the corrective force
when the pole's angular velocity is positive.
We also tighten the bias gap between the two actions (params[8] from
-1.8 to -1.5).  The difference of 0.5 still favours action 1 when
the pole tilts to the right, but the reduced bias makes the policy a
bit less aggressive, further preventing the cart from overshooting
early.
All other parameters are kept unchanged because they
already fall within the high-performance region identified in
previous trials.  The adjustments are small (at most 1.0)
and target only the aspects that appear linked
to the occasional failures , aiming to raise the mean reward closer
to the target of approximately 500 without harming the already good
median performance.
```

Despite this careful, well-justified reasoning, the revision destroys the policy ($436.05 \rightarrow 173.30$). This demonstrates that even minimal, statistically-informed edits can fail due to LLM hallucination in numerical parameter output, and why selection remains essential.

Loop Diagnosis (FrozenLake, run 2, episode 26)

Initial reward: 0.45 **Revised reward:** 0.70 $\Delta = +0.25$

Stats given to Critic-LLM:

```
Reward:  mean=0.45, min=0.00, max=1.00
Episode length:  mean=31.0, min=8, max=100
```

```
Success rate:  9/20 rollouts reached reward=1.00
Failure rate:  11/20 rollouts finished below reward=1.00
Median rollout (rollout 0, reward=0.0000, length=16,
outcome=terminated before the rollout cap)
```

Critic-LLM reasoning:

```
params[0]:  0, params[1]:  3, params[2]:  3, params[3]:  3,
params[4]:  0, params[5]:  0, params[6]:  0, params[7]:  0,
params[8]:  3, params[9]:  1, params[10]:  0, params[11]:  2,
params[12]:  1, params[13]:  2, params[14]:  1, params[15]:  3
I examined the failure pattern in the median rollout:
the agent repeatedly cycled between states 9 and 10,
never making progress toward state 15. This is caused by the
current action at state 10 (action 1) which chooses from {10→10,
10→11, 10→14}.  The 10→10 transition is a self-loop, and 10→11
leads straight into a hole.  Consequently, many episodes terminate
early due to this loop rather than stochasticity.
To mitigate this, I changed the action at state 10 to action
0 (down-left-up), which redistributes transitions to {10→14,
10→9, 10→6}.  This breaks the 9↔10 loop and provides a higher
probability of moving toward the goal (through state 14).  All
other state-action pairs remain unchanged because they already
contribute positively or do not manifest systematic failures, as
seen in earlier trials (e.g., trial 21 had a similar configuration
with a mean reward of 0.75).  This targeted change should lift the
mean reward toward the 1.0 target without altering well-behaving
parts of the policy.
```

K Custom environment specifications

To enable a direct comparison with prior work, we reuse three custom environments introduced by [29] (Maze, Nim, and Pong) without modification. We restate their specifications here for self-containment; the original definitions are due to [29].

Maze.

- **Description:** The agent traverses a 3×3 grid from a fixed start to a fixed goal. The grid contains impassable walls.
- **Observation space:** Discrete(9) – the agent's position.
- **Action space:** Discrete(4) – {Up, Down, Right, Left}.
- **Reward:** $+1$ for reaching the goal, -0.011 per step.
- **Termination:** Goal reached, or 100 steps elapsed.

Nim.

- **Description:** A game starting with 10 sticks. The agent and a rule-based optimal opponent alternate turns, removing 1–3 sticks each. The player forced to take the last stick loses. The agent moves first.
- **Observation space:** Box(11,) – integer count of remaining sticks.
- **Action space:** Discrete(3) – number of sticks to remove (1, 2, or 3).
- **Reward:** $+1$ for winning, -1 for losing.
- **Termination:** All sticks removed.

Pong.

- **Description:** The agent controls the left paddle. The right paddle is controlled by a rule-based optimal policy that never misses. The agent's goal is to return the ball as many times as possible.

- **Observation space:** Box(5,) – ball position (x, y), ball velocity (v_x, v_y), and agent paddle's y position.
- **Action space:** Discrete(3) – {Up, Stay, Down}.
- **Reward:** $+1$ each time the agent successfully hits the ball.
- **Termination:** The agent misses the ball, or 3 successful hits.

L R2PO Prompt Templates

We provide the prompts used by the Search-LLM and the Critic-LLM. The Search-LLM prompt closely follows the ProPS prompt design [29], which we adopt as the global-optimizer stage of our two-stage framework. Our novel contribution is the Critic-LLM prompt, which introduces a calibrated trajectory evidence package (median rollout, aggregate rollout statistics) and a revision rule controlled by τ_c that instructs the model to preserve already-strong policies. Separate variants are used for continuous (linear policy) and discrete (tabular policy) tasks. Placeholders in angle brackets (e.g., <RANK>, <HISTORY>) are filled at runtime.

The Critic-LLM prompt includes an environment description (<ENV_DESCRIPTION>). We use the same environment descriptions as [29] (Appendix L), with minor corrections to typos and, in some cases, to the policy parameterization (e.g., the stated number of parameters). The custom environments (Maze, Nim, and Pong) follow the specifications of [29], which we restate for self-containment in Appendix K.

L.1 Fixed-value placeholders

Two placeholders represent fixed optimization conventions held constant across all environments without per-task tuning:

- <STEP_SIZE> $= 1.0$. Used in the continuous-policy Search-LLM prompt as a suggested exploration step, and in the continuous-policy Critic-LLM prompt as a soft upper bound on per-parameter adjustments unless evidence justifies a larger change. Not used by the discrete-policy prompts.
- Continuous parameter range $[-6.0, 6.0]$ with one-decimal precision, hard-coded in both continuous prompts. This range comfortably contains the optimal parameter scales observed in preliminary runs across all continuous-state environments and is held fixed across all methods, including the ProPS and ProPS+ baselines, for matched-budget comparison.

L.2 Search-LLM Prompt (Continuous)

Search-LLM: Continuous Prompt

```
You are good global optimizer, helping me find the global maximum of a mathematical
function f(params).  I will give you the function evaluation and the
current iteration number at each step.  Your goal is to propose input values that
efficiently lead us to the global maximum within a limited number of iterations
(<MAX_ITERATIONS>).
# Regarding the parameters params:
params is an array of <RANK> float numbers.
params values are in the range of [-6.0, 6.0] with 1 decimal place.
# Here's how we'll interact:
1.  I will first provide MAX_STEPS (<MAX_ITERATIONS>) along with a few training examples.
2.  You will provide your response in the following exact format:
  * Line 1:  a new input 'params[0]:  , params[1]:  , ..., params[<RANK-1>]:'
    aiming to maximize the function's value f(params).
    Please propose params values in the range [-6.0, 6.0], with 1 decimal place.
  * Line 2:  detailed explanation of why you chose that input.
3.  I will then provide the function's value f(params) at that point,
  and the current iteration.
4.  We will repeat steps 2-3 until we reach the maximum number of iterations.
# Remember:
1.  Do not propose previously seen params.
2.  The global optimum should be around <OPTIMUM>.  If you are below that,
```

this is just a local optimum. You should explore instead of exploiting.
3. Search both positive and negative values. During exploration, use search step size of <STEP_SIZE>.

Previous optimization history
For reference, here are previously tried policies and their outcomes in a compact format.
Next, you will see examples of params and $f(\text{params})$ pairs.
<HISTORY>

Now you are at iteration <STEP_NUMBER> out of
<MAX_ITERATIONS>. Please provide the results in the indicated format.
Do not provide any additional texts.

L.3 Search-LLM Prompt (Discrete)

Search-LLM: Discrete Prompt

You are good global optimizer, helping me find the global maximum of a mathematical function $f(\text{params})$. I will give you the function evaluation and the current iteration number at each step. Your goal is to propose input values that efficiently lead us to the global maximum within a limited number of iterations (<MAX_ITERATIONS>).

Regarding the parameters params:
params is an array of <RANK> int numbers.
params values should be an integer chosen from <ACTIONS>.

Here's how we'll interact:
1. I will first provide MAX_STEPS (<MAX_ITERATIONS>) along with a few training examples.
2. You will provide your response in the following exact format:
   * Line 1: a new input 'params[0]: , params[1]: , ..., params[<RANK-1>]:' aiming to maximize the function's value $f(\text{params})$. Please propose params values from <ACTIONS>.
   * Line 2: detailed explanation of why you chose that input.
3. I will then provide the function's value $f(\text{params})$ at that point, and the current iteration.
4. We will repeat steps 2-3 until we reach the maximum number of iterations.

Remember:
1. Do not propose previously seen params.
2. The global optimum should be around <OPTIMUM>. If you are below that, this is just a local optimum. You should explore instead of exploiting.
3. Search all the possible values of params.

Previous optimization history
For reference, here are previously tried policies and their outcomes in a compact format.
Next, you will see examples of params and $f(\text{params})$ pairs.
<HISTORY>

Now you are at iteration <STEP_NUMBER> out of
<MAX_ITERATIONS>. Please provide the results in the indicated format.
Do not provide any additional texts.

L.4 Critic-LLM Prompt (Continuous)

Critic-LLM: Continuous Prompt

You are an optimization agent, tasked to optimize a linear policy for a reinforcement learning control task. An initial set of parameters was just proposed and tested.
Your job is to review, reflect on the test results and improve the policy parameters.

Task description
<ENV_DESCRIPTION>

Policy structure
The policy is a linear function with <RANK> floating-point parameters.
params values should be in the range [-6.0, 6.0] with 1 decimal place.
The episodic reward measures how well the policy performs.

What was proposed
The optimizer suggested these parameters:
<PROPOSED_PARAMS>

How it performed

```
Average reward over evaluation rollouts: <ACHIEVED_REWARD>
Target reward for this task: around <OPTIMUM>.

# What actually happened during the test
Below is a summary of <K> rollout statistics, along with one median rollout as a
representative example. <TRAJECTORY_SUMMARY>

# Previous attempts
For context, here are previous parameter settings and outcomes in a compact format.
<HISTORY_TEXT>

# Your task
Follow these steps before writing any parameters:
1. Use the reward and episode-length statistics to separate systematic failures
   from occasional ones.
2. Identify which state dimensions and action responses are associated with the
   representative median rollout.
3. Preserve behaviors that are consistent with the stronger aggregate statistics.
4. Revise only the parameters most responsible for instability or underperformance.

# Important points of consideration:
- Use the reward and length statistics to judge whether the policy is consistently
  weak or only occasionally failing.
- If success rate is already high, avoid overfitting to the representative trajectory.
- Use the median rollout as evidence of typical behavior.
- Keep changes focused.
- When achieved_reward >= <REVISION_THRESHOLD>, prefer zero
  changes or only one very small, strongly justified edit.
- Prefer adjustments of <STEP_SIZE> or less per parameter unless the
  evidence strongly supports a larger change.
Your response must have exactly two parts, in this order:

FIRST LINE (parameters only, nothing else before this):
params[0]: <value>, params[1]: <value>, ..., params[<RANK-1>]: <value>

REMAINING LINES (your reasoning):
Explain what you changed and why.
```

L.5 Critic-LLM Prompt (Discrete)

Critic-LLM: Discrete Prompt

```
You are an optimization agent, tasked to optimize a tabular policy for a
reinforcement learning task. An initial set of parameters was just proposed and
tested. Your job is to review, reflect on the test results and improve the policy
parameters.

# Task description
<ENV_DESCRIPTION>

# What is a policy here
The policy is an array of <RANK> integers. Each entry params[i]
says which action the agent takes in state i. The possible actions are
<ACTIONS>.

# What was proposed
The optimizer suggested the following policy:
<PROPOSED_PARAMS>

# How it performed
The policy was evaluated in the environment and achieved an average reward of
<ACHIEVED_REWARD>.
The target reward for this task is around <OPTIMUM>.

# What actually happened during the test
Below is a summary of <K> rollout statistics, along with one median rollout as a
representative example.
<TRAJECTORY_SUMMARY>

# Previous optimization history
For reference, here are previously tried policies and their outcomes in a compact format.
<HISTORY_TEXT>

# Your task
Follow these steps before writing any parameters:
1. Use the reward and episode-length statistics to identify whether failures are
```

```
  consistent versus occasional.
2.  Use the reward and episode-length statistics to judge whether failures are
  systematic or mostly due to stochasticity.
3.  Locate the state-action decisions in the policy that most likely explain the
  representative median rollout and the aggregate statistics.
4.  Revise only the parameters most responsible for the failures or missed
  opportunities.
# Important points of consideration:
- Use the reward and length statistics to distinguish consistent problems from
  occasional failures.
- If success rate is already high, avoid over-correcting based on a single trajectory.
- Use the median rollout as a representative example of typical behavior.
- Keep changes focused.  If part of the policy is working, preserve those parameters.
- When achieved_reward >= <REVISION_THRESHOLD>, prefer zero
  changes or only one very small, strongly justified edit.
- Remember that each parameter value must be chosen from <ACTIONS>.
Your response must have exactly two parts, in this order:
FIRST LINE (parameters only, nothing else before this):
params[0]:  <value>, params[1]:  <value>, ..., params[<RANK-1>]:  <value>
REMAINING LINES (your reasoning):
Explain what you changed and why.
```

M Dynamic ASV Debris Collection

We evaluate R2PO on a simulated dynamic autonomous surface vessel (ASV) debris-collection task, where a compact 14-parameter policy prioritizes drifting debris under a finite mission-time budget. Collecting an item consumes travel and handling time, while the remaining items continue drifting and may become unrecoverable. The policy must balance semantic priority, urgency, travel cost, and future collection opportunities.

We compare R2PO against CMA-ES, Random Search, ProPS+, and a distilled policy trained to imitate a model-based planner's action preferences. Each optimization method is allocated 200 candidate-policy evaluations, with 20 training missions per candidate, totaling 4,000 training missions. The best policy is selected using training return only, frozen, and evaluated on the same 20 held-out missions. Table 16 reports mean return and standard deviation across the held-out missions.

Table 16: Held-out return ($\pm$ standard deviation) across 20 evaluation missions. **Best** mean performance is highlighted.

Method	Held-out Return ↑
R2PO	$\mathbf{16.00 \pm 4.14}$
CMA-ES	15.85 ± 3.95
Random Search	15.30 ± 3.91
ProPS+	15.00 ± 3.67
Distilled Policy	14.85 ± 3.03

R2PO achieves the highest observed held-out mean return. The selected R2PO policy originates from a Critic revision that improved its corresponding Search proposal, illustrating the role of trajectory-grounded revision in this task.

N Robotic Table Tennis

We evaluate R2PO on the robotic table-tennis simulation task and structured motor-policy representation used by ProPS [29], based on the benchmark introduced by Celik et al. [2]. A seven-degree-of-freedom robotic arm is controlled using a Dynamic Motor Primitive (DMP) with 70 optimized parameters, comprising ten temporal basis weights per joint. The objective is to return an incoming ball toward a target at $(x, y) = (-0.6, -0.4)\,\mathrm{m}$ on the opponent's table.

We compare R2PO against ProPS+, CMA-ES, and PPO under a matched budget of 4,000 training rollouts. For the candidate-based optimizers, this corresponds to 200 candidate-policy evaluations

with 20 rollouts per candidate. The best policy is selected using training return only, frozen, and evaluated over the same 20 evaluation rollouts. Table 17 reports the resulting performance.

Table 17: Robotic table-tennis performance across 20 evaluation rollouts of one frozen policy per method. **Best** target performance is highlighted.

Method	Target Dist. (m) ↓	Legal Crossing	Table Landing	Target Success
R2PO	$\mathbf{0.132 \pm 0.000}$	100%	100%	**100%**
ProPS+	$\mathbf{0.132 \pm 0.000}$	100%	100%	**100%**
CMA-ES	0.263 ± 0.000	100%	100%	0%
PPO	0.338 ± 0.000	100%	100%	0%

R2PO successfully optimizes the structured 70-parameter motor policy, achieving a 100% target-success rate within the $0.20\,\mathrm{m}$ success radius, matching ProPS+ in final performance. Notably, R2PO first discovers a successful policy after 180 training rollouts, compared with 980 for ProPS+, requiring approximately $5.4\times$ fewer rollouts to reach the success criterion.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction present four contributions: the two-stage R2PO framework, systematic ablations, a mechanistic analysis of salience bias, and empirical validation. These are supported by Sections 3, 4.3, 4.4, and 4.2. R2PO matches or exceeds deep RL and prior LLM-based baselines on mean best reward across all 10 environments and achieves the highest mean reward on 9 of 10, with a statistical tie on the remaining environment. Additional comparisons with classical black-box optimizers and simulated robotic evaluations are reported in Section 4.2 and Appendices M and N.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Limitations are discussed in Section 5, including scope (compact policy classes only; deep neural network policies left as future work) and the conservative threshold τ_c, which requires environment-specific calibration (with sensitivity analysis in Appendix B).

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best

judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not present formal theoretical results. Mathematical expressions define the optimization objective, evaluation procedure, and algorithmic components, but are used to specify the method rather than to establish provable guarantees.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Section 4.1 specifies the ten benchmark environments, the LLM used (gpt-oss:20b via Ollama), the number of independent runs (10), rollouts per evaluation (K=20), iterations (T=100 for two-call methods, $2T$=200 for single-call methods), budget equalization protocol, SB3 baseline configuration, classical black-box optimizer settings, and conservative threshold values (Appendix F). The algorithm is fully specified in Algorithm 1 (Appendix E). The additional robotic tasks and evaluation protocols are described in Appendices M and N. Code for the core benchmark experiments is provided as supplementary material.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example

(a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
(b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The core R2PO implementation, experimental configurations, baseline scripts, and analysis tools are publicly available at `https://github.com/HWUDLabAIRoboticsResearch/R2PO`, together with installation and execution instructions.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 4.1 specifies the LLM model, number of independent runs, rollouts per evaluation, iteration counts, budget equalization, SB3 baseline hyperparameter protocol, classical black-box optimizer settings, and conservative threshold values. The methodology (Section 3) fully describes the algorithm. Threshold values per environment are in Appendix F. The additional simulated robotic tasks, training budgets, and held-out evaluation protocols are described in Appendices M and N.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: All results in Tables 1, 2, 3, 4, and 12 report mean $\pm$ standard deviation across 10 independent runs, as specified in Section 4.1. We further report Welch's t-tests with Holm correction for the core deep RL and LLM-based baselines in Appendix A.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The LLM (gpt-oss:20b) is served via Ollama on cloud GPU infrastructure (single GPU per inference instance). For the ten-environment benchmarks, each LLM-based method receives 200 LLM calls and 4,000 environment rollouts per run. SB3 baselines train on 4,000 episodes per run without LLM calls. Classical black-box optimizers require no LLM calls and use matched candidate-policy evaluation budgets ($N \in \{192, 196, 200\}$), with 20 rollouts per evaluation (Section 4.1). Environment rollouts use Gymnasium and MuJoCo simulators on CPU. The additional robotic experiments use the training budgets described in Appendices M and N.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research uses publicly available environments (Gymnasium, MuJoCo) and an open-source language model. No human subjects, private data, or sensitive applications are involved.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The paper studies LLM-based policy search in simulated reinforcement learning environments, with no direct deployment-stage applications. Positive impacts include improved interpretability of policy learning, since the LLM produces natural language reasoning alongside parameter updates. Potential negative impacts are limited given the compact-policy scope: as with any LLM-driven optimization, automation bias or unintended optimization toward poorly-specified objectives could occur if the method were applied to consequential real-world tasks without appropriate safeguards. We do not foresee direct paths to misuse from the methodological contributions in this paper.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release pretrained models, datasets, or any assets with high risk for misuse. The released code implements a policy search algorithm for simulated environments.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: Gymnasium [26] (MIT license), MuJoCo [25] (Apache 2.0), Stable-Baselines3 [20] (MIT license), and ProPS [29] are cited. The LLM used in all experiments is OpenAI's open-weight `gpt-oss-20b` model (released August 2025 under Apache 2.0; [18]), accessed via Ollama's cloud-hosted endpoint [17]. The classical black-box optimizers use `pycma` [8] and `scikit-optimize` [9], both licensed under BSD-3-Clause.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: The public R2PO repository provides the core algorithm implementation, configurations, prompts, and execution instructions.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.

- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: the paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLMs are the central component of the proposed method. Section 3 describes how two LLM calls (Search-LLM and Critic-LLM) are used at each iteration for policy proposal and trajectory-grounded revision. Section 4.1 identifies the specific model (gpt-oss:20b via Ollama).

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.