RLDISCOVER: LLM-DRIVEN CO-EVOLUTION OF REINFORCEMENT LEARNING ALGORITHMS

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM-guided program evolution has enabled discoveries in mathematics and computational optimization, raising the prospect of reinforcement learning (RL) algorithms that self-evolve to improve how agents learn. However, realizing this prospect faces two obstacles. Joint search over coupled algorithmic components is difficult to scale: simultaneous changes can disrupt learning, while isolated changes overlook their dependencies. Evaluating candidate algorithms also requires costly training, with fitness remaining uncertain across random seeds. We introduce RLDiscover, a framework for the self-evolution of model-free deep RL algorithms. Progressive Co-Evolution advances from targeted component edits to joint evolution, while Progressive Probabilistic Evaluation balances search breadth and evaluation fidelity through staged training and repeated evaluation. Experiments across SAC, PPO, and DQN on four benchmark suites show substantial improvements in mean return, with per-family median gains of 32%–84% and a peak return ratio of approximately $363\times$ over a near-zero baseline. These gains include transitions from failed learning to successful task completion, and improvements persist when evolution starts from stronger open-source implementations. On measured SAC locomotion runs, evaluation uses approximately one-fifteenth the estimated compute required to fully evaluate the same candidate pool. Remarkably, independent searches repeatedly discover interpretable combinations of adaptive robust losses, progress-dependent value targets, and running statistics, with selected programs transferring to unseen tasks. These findings point toward a broader role for self-evolution in AI: discovering interpretable algorithms that improve how agents learn.[1]

1 INTRODUCTION

LLM-guided program evolution makes executable algorithmic logic a direct object of search: language models propose code changes, and evaluation feedback guides their selection (Lehman et al., 2022). FunSearch demonstrated the potential of this approach for mathematical discovery (Romera-Paredes et al., 2024); AlphaEvolve extended it to larger programs and computational optimization tasks (Novikov et al., 2025); and ShinkaEvolve developed mechanisms for more sample-efficient program evolution (Lange et al., 2025). These advances motivate applying program evolution to the learning procedures that govern how artificial agents improve through experience. Reinforcement learning (RL) offers a natural target: an RL algorithm is itself a program specifying how experience is transformed into updates to an agent's predictions and behavior Oh et al. (2020). We focus on model-free deep RL, encompassing value-based methods such as DQN (Mnih et al., 2015), actor-critic methods such as PPO (Schulman et al., 2017), and SAC (Haarnoja et al., 2018). These algorithms specify how experience is converted into learning targets, value estimates, and policy updates. Their implementations therefore offer a structured space for discovering improvements to how agents learn.

However, evolving interacting components of deep RL algorithms faces two difficulties. First, *joint search over coupled components is difficult to scale*. Broad randomization often yields little learning, while the same design change can have opposite effects: increasing data reuse can help PPO but hurt vanilla policy gradients (Andrychowicz et al., 2021). These observations show that broad

[1] Code and data will be publicly released upon acceptance.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

simultaneous changes can disrupt learning, while optimizing choices in a fixed context can overlook their dependencies. Second, *candidate fitness is expensive and imperfectly observed*. In our experiments, fully evaluating 230 candidate programs would cost approximately 1330 GPU-hours per search run (Appendix J, Table 8).[2] Moreover, full-length training does not eliminate uncertainty caused by random seeds: On `mw-pick-place-v3`, the same evolved program obtains returns of 2.9, 2887.3, and 3.9 across three seeds (Appendix K). These observations expose competing demands on the evaluation budget: exploring many programs requires inexpensive feedback, while determining whether apparent improvements persist requires more extensive, repeated training.

This motivates RLDiscover, a framework that addresses these challenges through structured proposals and staged evaluation. Its search space consists of **five algorithmic components** within a working RL implementation.[3] **Progressive Co-Evolution (PCE)** first allocates mutations among individual components using a UCB-inspired rule, then permits joint revisions across components, allowing the search to build on targeted changes while exploring their interactions. **Progressive Probabilistic Evaluation (PPE)** allocates training compute through increasingly demanding evaluations, reserving full training for a small set of finalists. Together, they implement a simple design principle: structure the search to generate useful candidates, and increase evaluation effort as the candidate pool is narrowed. We instantiate the framework on SAC, PPO, and DQN, covering off-policy and on-policy learning as well as continuous and discrete control. Our contributions are:

1. **Efficient RL algorithm evolution.** We introduce PCE and PPE to address component dependencies and costly, noisy evaluations. On measured SAC locomotion runs, they reduce evaluation cost by approximately $15\times$ relative to evaluating the same number of candidates at full fidelity.
2. **Broad improvements across algorithms and benchmarks.** Across four benchmark suites, RLDiscover improves mean return on all 26 tested task–algorithm pairs spanning SAC, PPO, and DQN, with median gains of 39.8%, 83.6%, and 32.1%, respectively. Improvements also persist when evolution starts from stronger open-source SAC implementations, with positive gains on all nine tested tasks.
3. **Convergent discovery of interpretable mechanisms.** Task-specific searches repeatedly recover shared mechanisms, including robust critic losses in 16/17 SAC and 5/5 PPO champions. Cross-task evaluations further demonstrate transfer of evolved programs to unseen tasks, showing that program evolution recovers reusable learning mechanisms beyond hyperparameter tuning.

2 Related Work

Discovering RL algorithms. Prior work discovers learning objectives and update rules through evolutionary optimization and meta-learning. Evolved Policy Gradients (Houthooft et al., 2018) and MetaGenRL (Kirsch et al., 2020) learn parameterized objectives, while LPG(Oh et al., 2020) and DiscoRL (Oh et al., 2025) meta-learn neural update rules. AutoML-Zero and Evolving RL Algorithms search programs or computation graphs constructed from predefined operation libraries (Real et al., 2020; Co-Reyes et al., 2021). Eureka instead generates reward programs within an existing learning pipeline (Ma et al., 2024). A parallel line applies the same foundation-model code-evolution paradigm to adversarial multi-agent games, where the evaluation landscape shifts as strategies improve and reliable assessment therefore requires evaluator co-evolution and hierarchical evaluation (Anonymous, 2026). Our focus is source-code modification across multiple components of working RL implementations. Through fixed function interfaces, RLDiscover edits observation processing, targets, objectives, and reward shaping, and studies how to coordinate these modifications under a limited evaluation budget.

Multi-fidelity evaluation and search allocation. Successive Halving (Jamieson & Talwalkar, 2016) and Hyperband allocate (Li et al., 2018) increasing resources to promising configurations;

[2] The evaluation strategy introduced below uses approximately 87 GPU-hours for this candidate volume, a $15.3\times$ reduction relative to the estimated full-evaluation cost.

[3] The interfaces cover common RL computations: observation processing, target/advantage estimation, value-function losses, policy objectives, and reward processing with exploration bonuses (Bou et al., 2023; Burda et al., 2019). Their use is trainer-dependent. Appendix D gives the full interface specifications.

BOHB combines this allocation with model-guided candidate generation (Falkner et al., 2018). PPE applies staged evaluation to evolving RL programs to balance candidate coverage against the reliability of performance estimates. For proposal allocation, PCE uses a UCB-inspired rule to select components for mutation (Auer et al., 2002), before allowing joint program revisions. These mechanisms adapt established allocation ideas to the coupled search and evaluation of RL implementations.

3 METHOD

3.1 FIVE-COMPONENT DECOMPOSITION

Given a base RL algorithm, let $\mathbf{m} = (m_1, \ldots, m_K)$, with $K = 5$, denote the implementations of its five algorithmic components. Let $\mathcal{A}(\mathbf{m})$ be the algorithm obtained by inserting these components into the base implementation. For a fixed task environment e, let $\mathcal{S}$ be a finite set of training seeds and let s be sampled uniformly from $\mathcal{S}$. We seek to maximize

$$F(\mathbf{m}) = \mathbb{E}_{s\sim\mathrm{Unif}(\mathcal{S})}\left[R_e(\mathrm{Train}(\mathcal{A}(\mathbf{m}), e, s))\right], \tag{1}$$

where $\mathrm{Train}(\mathcal{A}, e, s)$ returns the policy obtained by training algorithm $\mathcal{A}$ in environment e with seed s under the full-fidelity training budget. For a policy π, $R_e(\pi)$ denotes its mean evaluation return in e, normalized using task-specific bounds. Thus, $F(\mathbf{m})$ measures the performance of the resulting algorithm. We write $\widehat{F}_\ell(\mathbf{m})$ for the observed fitness computed using the training budget.

3.2 PROGRESSIVE CO-EVOLUTION (PCE)

PCE searches for improvements in $F(\mathbf{m})$ by first restricting mutations to individual components and then allowing coordinated changes across components (Algorithm 1). This separates the search for useful local changes from the exploration of their interactions.

Phase 1: bandit-guided single-component evolution. Each component is an arm of a UCB1 bandit (Auer et al., 2002). Writing n_k for the number of pulls of arm k, $\bar{r}_k$ for its mean reward, t for the total pulls so far, and $\mathcal{K}_t$ for the arms still active, the next component to mutate is

$$k_t = \arg\max_{k\in\mathcal{K}_t}\left[\bar{r}_k + c_t\sqrt{\frac{\ln t}{n_k}}\right], \tag{2}$$

with K=5 arms and a decaying exploration coefficient $c_t = c_0/\sqrt{1 + t/K}$, c_0=$\sqrt{2}$ and c_t shifting the bandit toward exploitation as Phase 1 proceeds. For the selected component k, the LLM generates a batch of proposed programs $\mathbf{m}'$, each changing only m_k. These candidates are screened using $\widehat{F}_0(\mathbf{m}')$ (Section 3.3). The arm reward is the batch's nonnegative relative fitness gain over that component's incumbent. After initial coverage of all components, Phase 1 freezes components whose relative gains remain below a threshold for consecutive batches. It ends when all components are frozen or its budget cap is reached, returning per-component elites and their observed gains.

Phase 1 $\rightarrow$ 2 handoff. Component champions are merged in descending order of their Phase-1 gains. A proposed replacement $\mathbf{m}'$ is retained only if $\widehat{F}_\ell(\mathbf{m}') \geq \widehat{F}_\ell(\mathbf{m})$ under the same validation protocol. The resulting program initializes Phase 2. The insight library (below) is already populated from Phase 1 batches, arm-freeze comparisons, and rejected components, so Phase 2 starts from accumulated findings, negative ones included.

Phase 2: round-based full co-evolution. Phase 2 uses the remaining budget to pursue improvements in the same objective $F(\mathbf{m})$, with mutations now allowed to change any subset of components. Each round launches a fresh population-based search seeded with the current best program. Discoveries carry across rounds via an *insight library*, accumulated from Phase 1 onwards: after each round the mechanisms distinguishing that round's champions are extracted into short natural-language entries and injected into the next round's mutation prompt, so later rounds build on earlier findings.

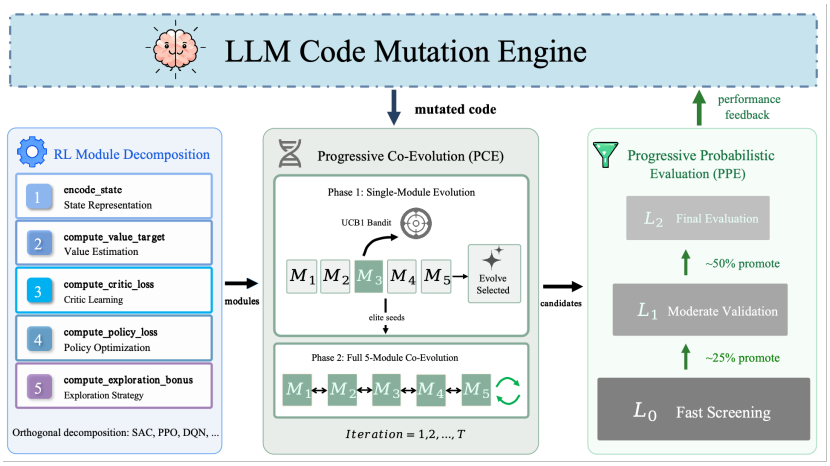


Figure 1: RLDiscover overview. **Left**: 5 algorithm components of RL algorithms (state representation, value estimation, critic learning, policy optimization, exploration). **Center**: PCE evolves components independently (Phase 1, UCB1 selection), then co-evolves them jointly (Phase 2). **Right**: PPE filters candidates through L0→L1→L2, so only candidates that survive cheap screening consume a full-length training run. **Top**: the LLM mutation engine generates code variants guided by performance feedback.

3.3 Progressive Probabilistic Evaluation (PPE)

PPE allocates evaluation compute to identify high-performing programs under the objective $F(\mathbf{m})$. Let $\widehat{F}_\ell(\mathbf{m})$ denote the observed fitness at fidelity ℓ. These observations differ from the full-training objective because of training truncation and seed-dependent variability. PPE screens candidates through three fidelity levels: **L0** (50K steps, one seed), **L1** (200K steps, two seeds), and **L2** (1M steps, three seeds). A single L2 evaluation costs 4–10 GPU-hours in our measurements. The resulting hierarchy supports broad candidate screening while reserving expensive validation for a smaller pool (Appendix D, Table 4).

Leader-referenced promotion. Promotion to L1 is governed by a global quota Q. Let $\mathcal{C}_r$ denote the candidates evaluated at L0 in round r, ordered by decreasing $\widehat{F}_0$ as $\mathbf{m}_{(1)}, \mathbf{m}_{(2)}, \ldots$, and let $\mathcal{P}$ be the accumulated L1 pool. Once $\mathcal{P}$ is nonempty, define

$$\mathbf{p}^\star \in \arg\max_{\mathbf{p}\in\mathcal{P}} \widehat{F}_1(\mathbf{p}). \tag{3}$$

The gate compares new candidates with the current higher-fidelity leader on the same low-fidelity scale. Round r promotes

$$\mathcal{C}_r^{\uparrow} = \{\mathbf{m}_{(1)}\} \cup \left\{\mathbf{m}_{(i)} : i > 1,\ \widehat{F}_0(\mathbf{m}_{(i)}) > \widehat{F}_0(\mathbf{p}^\star)\right\}, \tag{4}$$

truncated in L0 rank order to the remaining quota. The round's highest-scoring candidate therefore receives an L1 evaluation whenever quota remains, even if no candidate exceeds the gate. In the final rounds, an underspent quota triggers a switch to the pool's median L0 score as the promotion threshold.

Final selection. The five highest-scoring programs in $\mathcal{P}$ under $\widehat{F}_1$ are evaluated at L2. The reported program is the one with the highest $\widehat{F}_2$ among the audited L2 candidates. Ranking at every level uses untruncated normalized fitness. Before reporting, each champion is reloaded, checked for the required component functions, and compared with the seed program to detect silent fallbacks. AST-hash deduplication avoids repeated evaluation of syntactically identical programs. Prompt construction, syntax validation, and audit details are provided in Appendix D.

4 EXPERIMENTS

Tasks. SAC is evolved on 17 continuous control tasks—4 DMControl (Tassa et al., 2018), 5 Gymnasium (Towers et al., 2024), 5 Meta-World (Yu et al., 2020) and 3 MetaDrive (Li et al., 2023)—covering locomotion, manipulation and autonomous driving, with observation dimensions 11–348 and action dimensions 2–17; PPO is evolved on 5 Gymnasium locomotion tasks and DQN on 4 discrete control tasks (CartPole, MountainCar, Acrobot, LunarLander): 26 task–algorithm pairs in total. Per-task specifications are in Appendix G.

Two kinds of seed program. The *own-init* batch seeds evolution from our own standard implementation of the algorithm and covers all 26 pairs; the *SOTA-init* batch re-seeds the 9 SAC locomotion tasks that have an open-source reference—denisyarats/pytorch_sac (Yarats & Kostrikov, 2020) for DMControl, CleanRL SAC (Huang et al., 2022) for Gymnasium—from that reference at unchanged capacity and hyperparameters. Each batch is compared only against the baseline it was seeded from and neither filters out any task, so each improvement is attributable to evolution rather than to a change of implementation. Seed programs, hyperparameters, the candidate budget and the L2 evaluation protocol are in Appendix D.

4.1 RESULTS ACROSS THREE ALGORITHM FAMILIES

Table 1 reports every per-task L2 cell for all three instantiations. All three are evolved by the same pipeline—same algorithm-component decomposition, same PPE funnel, same L2 protocol—differing only in the seed implementation and component calling conventions, with DQN's policy-loss slot fixed rather than searched (Table 3). Evolution improves the baseline it was seeded from on **all 26 task–algorithm pairs**, with median gains of $+39.8\%$ on SAC, $+83.6\%$ on PPO and $+32.1\%$ on DQN, and it does so from one pipeline and one budget per task rather than per-task tuning. The gains span three orders of magnitude, from $+0.9\%$ to $\times 363$.

Search can reorder the algorithms themselves. PPO shows the largest median gain of the three families ($+83.6\%$), consistent with PPO's seed program having the lowest absolute starting point of the three families—more room to improve, larger gains. On gym-Hopper-v5, evolved PPO (3525.5 ± 57.8) ends above our SAC champion on that task (2780.7 ± 861.6), with a fifth of the seed spread—the weaker family evolved past the stronger one. Which family to start from is itself a search outcome, not something fixed in advance. The comparison holds because both numbers are raw returns on the same task under the same evaluation protocol; *fitness* is normalised per family and is not comparable across them, and neither is comparable to published numbers. Per-family qualifications, the † comparisons against external references and the normalization artifact behind MountainCar's high baseline fitness are in Appendix F.

Programs are diagnosable when search stalls. gym-Ant-v5 has the smallest own-init gain in the suite ($+1.4\%$). Reading the champion as code, rather than as a score, reveals why: three of the components are unchanged from the seed, and Ant's champion is the only one in the suite that kept plain MSE as the critic loss—the one change that virtually every other champion made (Appendix P). This points to insufficient search, not a task ceiling: the search did not explore enough from this starting point. The fix the diagnosis suggests is a stronger seed; re-seeding from CleanRL turns Ant's $+1.4\%$ into the largest gain in the SOTA-init block ($+27.3\%$). A fitness score cannot tell a hard task from an undersearched one; the program can.

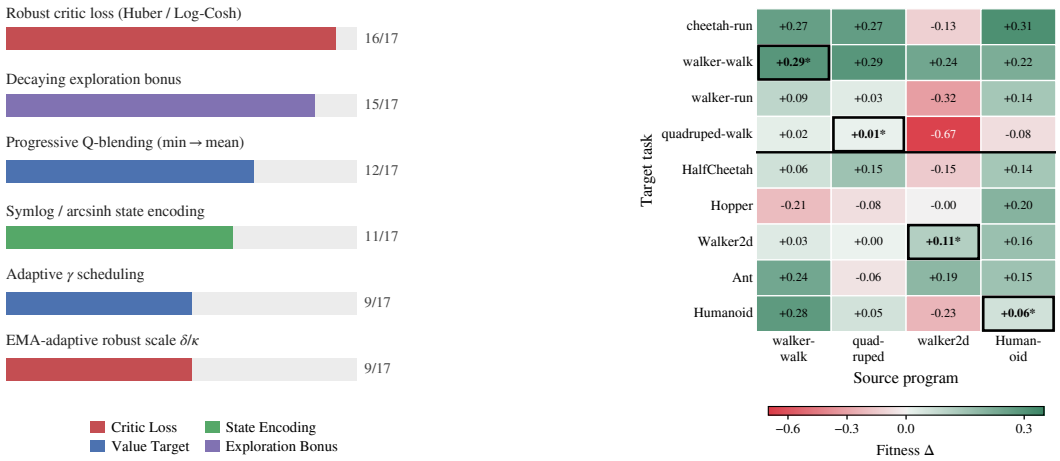
4.2 EVOLUTION FROM STRONGER SOTA BASELINES

The *SOTA-init* block of Table 1 addresses a natural concern: are the gains partly due to a weak starting implementation? The external baselines are indeed substantially stronger on several tasks (CleanRL reaches 11002.4 vs. our 5093.6 on HalfCheetah). Re-seeding from them, evolution improves on the SOTA starting point on **all 9 of 9** tasks, never degrades a strong seed, and finds double-digit gains on 3 (Ant $+27.3\%$, Hopper $+20.0\%$, cheetah-run $+20.0\%$).

The more informative quantity is what happens to the *gap* between the two batches, since that is what the concern is actually about. At the seed program the two implementations differ by a

Table 1: Full per-task results across the three algorithm families, all at L2 with 3 seeds $\{0, 42, 123\}$ (protocol in Appendix D; MountainCar-v0 is a 10-seed exception). Every block compares an evolved program against the baseline it was seeded from: our own SAC baseline (17 tasks), an open-source SOTA seed on all 9 locomotion tasks with an external reference (Section 4.2; no task is filtered out), and the standard PPO / DQN implementation of our framework. The SOTA-init block re-evolves the *same* 9 locomotion tasks and is not counted again in the 26. Fitness is the task-normalized return; Improvement is computed on raw returns; bold = improvement. Markers †, § and ¶ are explained in Appendix F.

Task	Init	Baseline Return	Evolved Return	Fitness	Improvement
SAC, own-init (17 tasks): evolved vs. our own SAC baseline; evolved § — median +39.8%					
dm-cheetah-run	Ours	302.3 ±37.3	646.9 ±67.4	0.647	**+114.0%**
dm-walker-walk	Ours	679.6 ±41.0	971.0 ±3.3†	0.971	**+42.9%**
dm-walker-run	Ours	487.7 ±56.3	585.7 ±7.1	0.586	**+20.1%**
dm-quadruped-walk	Ours	931.6 ±12.5	939.8 ±4.1†	0.940	**+0.9%**
gym-HalfCheetah-v5	Ours	5093.6 ±209.3	9053.7 ±1554.1	0.604	**+77.7%**
gym-Hopper-v5	Ours	2475.0 ±613.7	2780.7 ±861.6†	0.695	**+12.4%**
gym-Walker2d-v5	Ours	3883.1 ±218.3	5429.5 ±485.3†	0.905	**+39.8%**
gym-Humanoid-v5	Ours	4998.8 ±205.5	5281.2 ±152.7†	0.755	**+5.6%**
gym-Ant-v5	Ours	3732.8 ±516.1	3785.9 ±821.9	0.473	**+1.4%**
mw-push-v3	Ours	5.0 ±0.4	1825.1 ±1875.4	0.406	**×363**
mw-drawer-open-v3	Ours	3177.0 ±1190.9	4621.3 ±54.9	1.000	**+45.5%**
mw-door-open-v3	Ours	2906.0 ±889.8	3099.9 ±1923.0	0.689	**+6.7%**
mw-peg-insert-side-v3	Ours	1151.9 ±1607.1	3075.6 ±2172.8	0.683	**+167.0%**
mw-pick-place-v3	Ours	6.8 ±2.4	964.7 ±1359.5	0.214	**×141**
md-validation	Ours	147.8 ±85.8	323.2 ±150.3	0.923	**+118.7%**
md-10env	Ours	229.6 ±57.6	232.7 ±27.1	0.665	**+1.4%**
md-100envs	Ours	237.1 ±60.1	282.2 ±22.4	0.806	**+19.0%**
SAC, SOTA-init (9 tasks, not counted again): evolved vs. denisyarats / CleanRL seed; evolved § — median +2.6%					
dm-cheetah-run	denisyarats	713.3 ±140.1	855.8 ±33.7	0.856	**+20.0%**
dm-walker-walk	denisyarats	967.5 ±1.3	972.8 ±3.6	0.973	**+0.5%**
dm-walker-run	denisyarats	753.0 ±7.7	772.5 ±4.2	0.773	**+2.6%**
dm-quadruped-walk	denisyarats	924.9 ±14.1	940.8 ±17.3	0.941	**+1.7%**
gym-HalfCheetah-v5	CleanRL	11002.4 ±91.5	11064.9 ±240.2	0.738	**+0.6%**
gym-Hopper-v5	CleanRL	2360.8 ±624.8	2832.5 ±520.3	0.708	**+20.0%**
gym-Walker2d-v5	CleanRL	4672.3 ±354.0	4689.8 ±208.6	0.782	**+0.4%**
gym-Ant-v5	CleanRL	4074.7 ±793.5	5188.8 ±242.9	0.649	**+27.3%**
gym-Humanoid-v5	CleanRL	5165.5 ±414.1	5643.7 ±297.3	0.806	**+9.3%**
PPO (5 tasks): evolved vs. standard PPO baseline; evolved § — median +83.6%					
gym-Hopper-v5	Ours	1920.0 ±289.9	3525.5 ±57.8	0.881	**+83.6%**
gym-HalfCheetah-v5	Ours	1724.4 ±39.6	4233.6 ±349.8	0.282	**+145.5%**
gym-Ant-v5	Ours	776.0 ±76.3	2789.0 ±445.0	0.349	**+259.4%**
gym-Walker2d-v5	Ours	4023.4 ±655.0	4440.2 ±478.3	0.740	**+10.4%**
gym-Humanoid-v5	Ours	606.6 ±55.1	809.1 ±91.7	0.116	**+33.4%**
DQN (4 tasks): evolved vs. standard DQN baseline; evolved § — median +32.1%					
CartPole-v1	Ours	330.2 ±18.1	451.2 ±69.0	0.902	**+36.6%**
MountainCar-v0¶	Ours	−200.0 ±0.0	−144.9 ±29.2	0.937	**+27.5%**
LunarLander-v3	Ours	66.9 ±87.0	174.6 ±24.2	0.582	**+161.0%**
Acrobot-v1	Ours	−91.2 ±5.3	−87.0 ±6.6	0.983	**+4.7%**
26/26 audited improvements across the three families (17 SAC own-init, 5 PPO, 4 DQN)					



(a) Consensus innovations (17 champions)

(b) Cross-task transfer of 4 champions

Figure 2: Evolution finds algorithmic improvements rather than per-task overfits. **(a)** Frequency of independently discovered innovations across all 17 champions, coloured by the component each modifies. **(b)** Fitness Δ from running a source champion (column) on a target task (row); green = improvement, red = regression; boxed cells marked $*$ are the source task itself. Criteria, the full 17×6 matrix and per-task values: Appendices I and H.

mean of 23.4% in return; after evolution the mean gap is 13.1%, narrowing on 7 of the 9 tasks—on dm-walker-walk almost entirely ($29.8\% \rightarrow 0.2\%$). Evolution recovers roughly half of an implementation-quality gap it did not create, neither closing nor widening it. The two tasks where the gap instead *widens* are Humanoid and Ant, the latter being the same cell identified above as the one where own-init search failed: a stronger seed helps most exactly where the search itself did least.

Narrowing is not the same as converging, however, and the head-to-head outcome is a separate question. On the nine locomotion tasks—the only ones both batches were evolved on with 3 seeds—the SOTA-init champion finishes ahead on 8 of 9, losing only on Walker2d; two of those eight are ties of 0.2% or less, so the honest reading is six clear wins, one loss, and two ties. The gap narrows on 7 of 9 tasks while the ranking survives on only 4, and where it flips the per-task search gain rather than the seed decides it: Walker2d is a genuine overtake, Ant the mirror image. A large seed advantage is durable and a small one is not—all four preserved rankings sit on seed gaps of at least 30%. Per-task seed gaps, final gaps, and each batch's gain over its seed are in Appendix 5. The core innovations (Huber loss, Q-annealing) emerge in both batches—what we would expect of improvements to the algorithm itself rather than artifacts of a particular starting implementation.

4.3 Discovered Algorithmic Innovations

Independent evolution repeatedly converges on a small set of core modifications (Figure 2a), and the programs carrying them transfer to tasks they were never evolved on (Figure 2b); we take each task's best champion across both seed batches, covering all 17 tasks. Reading the PPO and DQN champions the same way (Appendix P.1) isolates exactly one innovation shared by all three families—replacing MSE with a robust regression critic loss, found in 5/5 PPO and 3/4 DQN champions alongside SAC's 16/17, i.e. 8 of the 9 cross-algorithm champions under two different trainers. Each family's *other* innovations sit in machinery specific to that family: GAE-λ scheduling and clipped-surrogate/KL control in 5/5 PPO champions, Q-annealing in SAC, which PPO cannot express because it has no twin critics. DQN is the honest exception: only three of its five slots are live under our trainer (the policy slot is fixed and the value target reduces to the plain $\max_a Q'$ backup), so its champions concentrate on encoding and exploration by construction rather than by discovery, with added critic-loss changes in three (Appendix P.1).

Robust critic loss. addresses bootstrapping noise: MSE amplifies large TD errors quadratically while robust losses cap gradients. All but one champion adopt one (Huber, smooth-L1, Log-Cosh, or Charbonnier) and 9 adapt the transition scale via an EMA of running error statistics. **Progressive Q-blending** replaces the fixed pessimistic $\min(Q_1, Q_2)$ inherited from clipped double-Q learning (Fujimoto et al., 2018) with a progress-annealed blend of $\min$ and mean—a schedule never previously proposed for SAC—with the annealing direction splitting by domain. **Symlog encoding** compresses observations via $\text{sign}(x) \cdot \log(1+|x|)$; a similar transform was independently proposed in DreamerV3 (Hafner et al., 2023). These modifications transfer (Figure 2b): of four champions re-evaluated on 9 locomotion targets, three improve 6–7 of their 8 non-source tasks (mean fitness Δ $+0.083$ to $+0.157$), while the walker2d program—which relies on hard-coded step thresholds—wins only 2/8 (-0.134; Appendices H, I).

Cross-domain generality. In Meta-World, push and pick-place achieve order-of-magnitude gains from near-zero baselines, demonstrating that evolution can fundamentally restructure failing training procedures. In MetaDrive, all 3 driving scenarios improve, including 100-envs large-scale generalization (+19.0%). The same evolution pipeline and evaluation protocol work across locomotion, robotic manipulation, and autonomous driving without domain-specific adjustments.

Interpreting the discovered mechanisms. Progressive Q-blending's annealing direction splits by domain: locomotion champions start optimistic and anneal toward $\min$, while manipulation and driving champions *relax* pessimism as training stabilizes. Symlog encoding prevents gradient saturation in high-dimensional tasks; MSE amplifies large TD errors quadratically while robust losses cap gradients at the transition scale, acting as built-in gradient clipping.

Principles for generalizable programs. Comparing a well-generalizing program (quadruped, 6/8 wins) with the worst (walker2d, 2/8 wins), we identify three principles for transferable RL modifications. **(1) Relative scheduling**: use progress ratio t/T rather than absolute step counts for annealing—the quadruped program's cosine-annealed schedules adapt to any training length. **(2) Stateless encoding**: pointwise transformations (arcsinh, symlog) transfer perfectly; running statistics do not. **(3) Adaptive thresholds**: parameters derived from training dynamics (EMA of TD errors, as in the walker-walk program) auto-calibrate to each task, while hard-coded constants may be inappropriate for different reward scales.

5 Ablation Studies

We ablate PCE+PPE with 3 groups of 8 tasks[4] under the matched-budget protocol of Appendix D. We compare the full method against five ablations. Two remove one PCE component at a time and keep everything else: *w/o component decomposition* keeps Phase 1 and the funnel but makes every mutation rewrite the whole program, and *w/o Phase 1* keeps the decomposition and the funnel but skips the bandit stage. The remaining three—*L0*, *L1* and *L2*—ask what a fixed evaluation fidelity is worth, and to answer that they run the search itself in its simplest form: the whole program is mutated as one block, Phase 1 is skipped, and every candidate is scored at that single level, with the compute-aligned budget the level allows. They therefore vary *three* factors against the full method at once (funnel, decomposition, Phase 1), and a gap between the full method and any of them cannot be attributed to the funnel alone. We read them as bounding what a single fidelity level can achieve rather than as isolating PPE.

The full method leads every ablation on the pooled mean. It leads all five by 0.026–0.043 normalised fitness over the 8 pairs and by 0.041–0.062 over the 6 non-degenerate pairs, and the sign of the paired Δ is negative for every ablation in both groupings. The per-pair spread is 1.3–$4\times$ the gap being measured, however, so the consistent ordering rests on the sign pattern rather than on any single comparison, and we read the pooled numbers as directional.

[4]The detailed configurations are: (1) SAC on `dm-cheetah-run`, `gym-HalfCheetah-v5`, `md-100envs`, and `mw-drawer-open-v3`; (2) The four SAC pairs in I, PPO on `gym-Hopper-v5` and `gym-HalfCheetah-v5`; and DQN on `LunarLander-v3` and `MountainCar-v0`; (3) The four SAC pairs in I, PPO on `gym-Hopper-v5`; and DQN on `LunarLander-v3`.

Table 2: Component ablation across the three algorithm families: audited L2 champion fitness, mean $\pm$ cross-task std over **3 independent evolution runs** per cell (8 pairs $\times$ 6 conditions $\times$ 3 runs $= 144$ runs); per-task matrix in Table 10, per-cell evolution spreads in Appendix N. The $\pm$ is the standard deviation *across tasks* within each row.

		Ablating PCE		Single-fidelity PPE		
	Full	w/o decomp.	w/o Phase 1	L0	L2	L1
(1)	**0.784** $\pm$0.09	0.762 $\pm$0.13	0.758 $\pm$0.14	0.728 $\pm$0.17	0.700 $\pm$0.08	0.691 $\pm$0.05
(2)	**0.731** $\pm$0.23	0.687 $\pm$0.25	0.704 $\pm$0.24	0.687 $\pm$0.25	0.700 $\pm$0.21	0.694 $\pm$0.22
(3)	**0.789** $\pm$0.08	0.731 $\pm$0.17	0.748 $\pm$0.13	0.727 $\pm$0.15	0.728 $\pm$0.10	0.736 $\pm$0.08

Stratifying by family shows why, and recovers the SAC result. The pooled null averages over families that disagree. Within SAC the funnel effect is large and ordered: on the four-task mean the full method leads all three single-level variants by 7.1% (L0), 10.7% (L2) and 11.8% (L1), with distinct failure modes—L0 searches the widest but ranks on 50K-step single-seed returns that favour fast starters not all of which survive to 1M steps; L2 ranks on the true objective but at $\sim$5.8 GPU-hours per evaluation affords only 13 candidates, degenerating into local refinement of a handful of programs; L1 inherits neither L0's breadth nor L2's fidelity. On PPO and DQN the directions are inconsistent or reversed—L2 beats the full method on *both* PPO tasks and wins 4 of the 8 pairs, two of them starred. Since the funnel's thresholds were calibrated on SAC, we claim the funnel advantage for SAC only.

Two statements hold in every family. **(a)** Single-level selection at the cheapest fidelity (L_0, the simplest search configuration) is the only ablation trailing the full method in every family. **(b)** Compute-aligned L2 evaluation is the closest competitor pooled across families, reaching comparable fitness by spending the whole budget on 13 fully-trained candidates and forgoing the funnel's wider search and early-stopping speedup. What both bound is the same trade-off: across the three configurations the search gets wider as the fidelity gets cheaper and the winner sits in between. Because the three single-level conditions also drop the decomposition and Phase 1, the full method's margin is evidence for the *combination*, and the defensible claim about the funnel specifically is *efficiency at equal quality* rather than demonstrably higher fitness.

Selection fidelity pays off where the search problem is hard. Separating six configurations requires headroom: if the baseline already saturates a cell, all six cluster on the same score and the ordering becomes noise. The four SAC pairs split by baseline strength (Appendix N, Table 11). Where the initial solution is weak (on dm-cheetah-run and gym-HalfCheetah-v5), even the *worst* ablated condition beats it by $+0.25$ and $+0.31$, and the full method ranks first of six on both. Where the initial solution already sits near the top of the range (on md-100envs and mw-drawer-open-v3), the worst condition clears it by only $+0.051$ and $+0.012$, the full method drops to fourth and second, and on md-100envs the winner is L0, the *cheapest* condition. These results suggest that selection fidelity matters most when the baseline leaves substantial room for improvement; benchmarks dominated by near-saturated baselines may therefore underestimate it.

6 Conclusion

RLDiscover advances automated RL algorithm evolution by searching executable learning rules under a limited evaluation budget. Progressive Co-Evolution coordinates edits to interacting components, while Progressive Probabilistic Evaluation concentrates training compute on promising candidates. Across SAC, PPO, and DQN, the evolved programs improve learning performance, including successful task completion where the starting implementation fails. The search uncovers interpretable combinations of adaptive robust losses, progress-dependent value targets, and stateful schedules, whose recurrence across independent searches and observed cross-task transfer provide evidence of reusable algorithmic structure. By connecting LLM-generated code with training-based selection, RLDiscover offers the community both a practical discovery framework and executable mechanisms to inspect, test, and adapt—evidence that program evolution can expand what agents learn and generate concrete insights for designing better RL algorithms.

AI USE STATEMENT

Large language models are the central experimental apparatus of this work, not an authoring aid. All candidate programs reported here—every evolved component implementation, including all champion programs analysed in Section 4.3 and Appendix I—were generated by an LLM acting as the mutation operator inside RLDiscover, under the prompting protocol and feedback format described in Section 3 and Appendix D. The identity of the mutation model is withheld for anonymity and will be stated in the camera-ready version together with decoding settings and the exact prompt templates; the generated programs themselves will be released with the code. Every program that enters a reported result was executed and scored by the automated evaluation pipeline, and all champion programs were additionally read and audited by the authors for degenerate behaviour (reward hacking, evaluation-protocol violation, or hyperparameter changes outside the component interface); the audit criteria and outcomes are given in Appendix I.

Separately, the authors used an LLM assistant for routine writing support—copy-editing, LaTeX table formatting, and plotting-script boilerplate. The authors verified every number in the paper against the raw experiment logs and take full responsibility for the content.

REPRODUCIBILITY STATEMENT

Section 4 specifies the benchmarks, training budgets, seed sets, and the L2 evaluation protocol used for every reported number; Appendix D gives the component interfaces, the PCE schedule, and the PPE promotion rule (Eq. 4). Per-seed results for all 17 SAC own-init cells and all 9 PPO/DQN cells are reported in Appendix K rather than summarised; the 9 SOTA-init cells are covered by the per-cell $\pm$ in Table 1 whose provenance is stated in that caption, and Section A states explicitly where our sample size (n=3 per cell) supports only qualitative claims. External SOTA reference values are re-run by us under our own evaluation protocol rather than copied from the cited papers. The evolution framework, the evolved champion programs, the evaluation harness, and the per-cell result data will be released publicly upon acceptance of the paper.

REFERENCES

Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, Raphaël Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. What matters in on-policy reinforcement learning? a large-scale empirical study. In *International Conference on Learning Representations*, 2021.

Anonymous. Beyond static evaluation: Co-evolutionary mechanisms for LLM-driven strategy evolution in adversarial games. *arXiv preprint arXiv:2606.10389*, 2026.

Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256, 2002.

Albert Bou, Matteo Bettini, Sebastian Dittert, Vikash Kumar, Shagun Sodhani, Xiaomeng Yang, Gianni De Fabritiis, and Vincent Moens. TorchRL: A data-driven deep reinforcement learning library as an out-of-the-box tool, 2023.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *International Conference on Learning Representations*, 2019.

John D Co-Reyes, Yingjie Miao, Daiyi Peng, Esteban Real, Quoc V Le, Sergey Levine, Honglak Lee, and Aleksandra Faust. Evolving reinforcement learning algorithms. In *International Conference on Learning Representations*, 2021.

Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*, 2018.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, 2018.

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*, pp. 1861–1870. PMLR, 2018.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

Rein Houthooft, Richard Y Chen, Phillip Isola, Bradly C Stadie, Filip Wolski, Jonathan Ho, and Pieter Abbeel. Evolved policy gradients. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G M Araújo. CleanRL: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.

Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *International Conference on Artificial Intelligence and Statistics*, 2016.

Louis Kirsch, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Improving generalization in meta reinforcement learning using learned objectives. In *International Conference on Learning Representations*, 2020.

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. ShinkaEvolve: Towards open-ended and sample-efficient program evolution. *arXiv preprint arXiv:2509.19349*, 2025.

Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O Stanley. Evolution through large models. In *Handbook of Evolutionary Machine Intelligence*. Springer, 2022.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.

Quanyi Li, Zhenghao Peng, Lan Feng, Qihang Zhang, Zhenghai Xue, and Bolei Zhou. MetaDrive: Composing diverse driving scenarios for generalizable reinforcement learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(3):3461–3475, 2023.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. In *International Conference on Learning Representations*, 2024.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J R Ruiz, Abbas Mehrabian, M Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

Junhyuk Oh, Matteo Hessel, Wojciech M Czarnecki, Zhongwen Xu, Hado van Hasselt, Satinder Singh, and David Silver. Discovering reinforcement learning algorithms. In *Advances in Neural Information Processing Systems*, 2020.

Junhyuk Oh, Gregory Farquhar, Iurii Kemaev, Dan A Calian, Matteo Hessel, Luisa Zintgraf, Satinder Singh, Hado van Hasselt, and David Silver. Discovering state-of-the-art reinforcement learning algorithms. *Nature*, 648(8093):312–319, 2025.

Esteban Real, Chen Liang, David R So, and Quoc V Le. AutoML-Zero: Evolving machine learning algorithms from scratch. In *International Conference on Machine Learning*, 2020.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco J R Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

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.

Denis Yarats and Ilya Kostrikov. Soft actor-critic (SAC) implementation in PyTorch. https://github.com/denisyarats/pytorch_sac, 2020.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-World: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning*, pp. 1094–1100. PMLR, 2020.

A Limitations

Three limitations bound what we claim. First, *PPO and DQN results are suboptimal relative to what per-algorithm tuning would achieve*: budgets and hyperparameters were carried over from SAC without adjustment, so the absolute returns for these two families are not comparable to published results and represent a lower bound on what the same pipeline could reach with algorithm-specific calibration.

Second, *the five-component decomposition is active only at L0*. Once a candidate advances to L1 or L2, it is evaluated as a monolithic program and the component interface plays no role. This confines Phase 1's per-component credit assignment, and any further structured exploration, to the 50K-step screening stage. A natural extension is to run L1 for a brief period and then re-enter L0 with the accumulated fitness signal—keeping structured five-component search active at longer horizons rather than switching permanently to monolithic evaluation.

Third, *L0 screening at 50K steps can discard RL strategies with slow warm-up*. Programs that achieve strong asymptotic performance but require extended environment interaction to stabilise are ranked low at the short horizon and filtered out before the costlier evaluations that would reveal their advantage. The funnel's promotion thresholds are calibrated to the evaluation protocol, not to the learning dynamics of each candidate.

B Algorithm–Component Mapping

Table 3 shows how standard model-free algorithms populate the five slots of Section 3.1, supporting the claim that the decomposition is a template rather than a SAC-specific factoring.

Table 3: Standard model-free algorithms populate the same five slots; the decomposition is a template rather than a SAC-specific factoring. *For value-based methods the policy slot is *degenerate*, not absent: the policy is the greedy $\arg\max$ over Q, so the slot is fixed rather than searched (we therefore exclude it from the search for DQN). We evolve the three families marked in bold.

Algorithm	encode	value target	critic loss	policy loss	exploration
DQN	MLP	TD target	MSE	greedy*	ϵ-greedy
Rainbow	MLP	n-step	distributional	greedy*	NoisyNet
DDPG / TD3	MLP	twin TD target	MSE	DPG	OU / Gaussian
A2C / **PPO**	MLP	GAE	MSE	clipped surrogate	entropy
SAC	MLP	soft Bellman	MSE	reparam. + entropy	entropy

C Implications for RL Algorithm Design (extended)

Our results suggest the standard SAC algorithm contains several recurring suboptimalities fixable across the majority of tasks tested: MSE is suboptimal for critic training (robust losses better on 16/17 tasks); fixed pessimistic Q-aggregation is overly conservative (annealing helps); raw observations benefit from logarithmic compression (symlog in 11/17); and constant-coefficient exploration bonuses are worse than progress-decayed ones. The approach holds across locomotion, manipulation (Meta-World 5/5), and driving (MetaDrive 3/3), consistent with cross-domain generality, and the PPO (5/5) and DQN (4/4) results obtained through the same five slots (Section 4.1) indicate the same holds across algorithm families.

Cross-domain generality. In Meta-World, push and pick-place achieve order-of-magnitude gains from near-zero baselines, demonstrating that evolution can fundamentally restructure failing training procedures. In MetaDrive, all 3 driving scenarios improve, including 100-envs large-scale generalization (+19.0%). The same evolution pipeline and evaluation protocol work across locomotion, robotic manipulation, and autonomous driving without domain-specific adjustments.

Cross-algorithm takeaway. Across three algorithm families spanning off-policy and on-policy, continuous and discrete control, RLDiscover improved the baseline on 26/26 tasks with audited, non-degenerate champions. Its two assumptions—that an RL algorithm decomposes into evolvable components, and that fitness can be estimated through a staged funnel—hold on all three, although the three are not validated to equal depth (Section A).

Per-task failure modes of the single-level PPE variants. L0-only costs -0.129 on dm-cheetah-run and -0.062 on mw-drawer-open; all-L2 costs -0.153 on md-100envs, the task with the most headroom.

Compositionality of the discovered innovations. Huber loss appears in DQN (Mnih et al., 2015) and symlog in DreamerV3 (Hafner et al., 2023), so the individual ingredients are not new. Evolution combines them with elements absent from prior work (adaptive δ from running statistics, asymmetric weighting from Q-disagreement), and the specific schedules (3-stage Q-blending with sinusoidal interpolation) are novel.

Why the consensus innovations work. Progressive Q-blending's annealing direction splits by domain: locomotion champions start optimistic and anneal toward $\min$, while manipulation and driving champions *relax* pessimism as training stabilizes. Symlog encoding prevents gradient saturation in high-dimensional tasks.

D Implementation Details of PCE and PPE

The five component interfaces. Each component is a Python function with a fixed calling convention, receiving the current step t and total steps T and sharing a persistent state dictionary; full Python signatures are included in the released code:

Algorithm 1 Progressive Co-Evolution (PCE)

Require: Base program p_0, components $\{m_1, \ldots, m_K\}$, candidate budget B (240), batch size b (10), L1 quota Q (30)

Phase 1: bandit-guided single-component evolution
while some arm is unfrozen **and** Phase-1 spend $<$ upper bound **do**
  Select arm $k^* = \arg\max_{k \in \mathcal{K}_t} \mathrm{UCB}(k)$ (Eq. 2); generate batch of b mutations of m_{k^*}; screen at L0
  Update bandit statistics $(\bar{r}_{k^*}, n_{k^*})$ with the batch's relative fitness gain
  Freeze arm k^* if κ consecutive pulls yielded relative gain $< 2\%$
end while
Greedily merge per-component champions into p_{merge}, keeping a component only if fitness does not regress
Phase 2: full co-evolution from p_{merge} with the remaining budget $B -$ (Phase-1 spend)
while budget remains **do**
  Run b full-program mutations seeded with the current best program and the insight library
  Screen at L0; promote top-1 plus all candidates clearing gate γ to L1 (Eq. 4, within quota Q)
  Extract this round's distinguishing mechanisms into the insight library
end while
Evaluate the L1 pool's top-5 at L2
return Best program by audited L2 score

1. **`encode_state`**: *State representation* — transforms raw observations into features (e.g., normalization, running statistics, dimensionality reduction).
2. **`compute_value_target`**: *Value estimation* — computes bootstrap targets for the critic. In SAC this is the soft Bellman target $\min(Q_1, Q_2) - \alpha \log \pi$; in PPO it is Generalized Advantage Estimation (GAE); in DQN the target-network Bellman backup.
3. **`compute_critic_loss`**: *Critic learning* — defines the loss for training value or Q-functions (e.g., MSE, Huber, value clipping).
4. **`compute_policy_loss`**: *Policy optimization* — computes the gradient signal for the policy. SAC uses reparameterized entropy-regularized objectives; PPO uses the clipped surrogate; DQN derives the policy implicitly via $\arg\max_a Q$.
5. **`compute_exploration_bonus`**: *Exploration strategy* — shapes the reward signal to encourage exploration (e.g., intrinsic curiosity, entropy bonuses, reward normalization).

Algorithm 1 gives the full PCE procedure summarized in Section 3.2, and Table 4 the three PPE fidelity levels of Section 3.3.

Table 4: PPE evaluation levels. Only programs passing lower levels are promoted.

Level	Steps	Seeds	Eval Eps	Measured wall-clock	Purpose
L0	50K	1	5	8–11 min (9.3)	Rapid screening
L1	200K	2	10	60–79 min (78)	Moderate validation
L2	1M	3	20	4.1–9.6 h (5.0)	Final confirmation

Evolution budget. All mutations are generated by a large language model. Every run in this paper—main-line and ablation alike—draws from a budget of B=240 candidate programs. After syntactic validation and AST-hash deduplication, each run screens on average **230 candidates** at L0, for ∼87 GPU-hours of evaluation per run (Appendix J). The ablations of Section 5 use the full procedure of Section 3.2 throughout, so all conditions compared there are internally consistent.

Matched-budget protocol for controlled comparisons. The ablations of Section 5 compare conditions under a matched mutation budget: every L0-screening condition receives 240 LLM-generated candidate programs, with the Phase-1/Phase-2 split determined adaptively as in Section 3.2; conditions that remove a phase spend the same 240 candidates in the remaining phase.

Conditions that evaluate every candidate at higher fidelity (L2-only, L1-only) cannot match candidate counts by construction, and are instead aligned by evaluation compute (same wall-clock GPU budget), which affords 13 (SAC), 43 (PPO) and 33 (DQN) L2-only candidates and 60 L1-only ones (realized ≈50 for L1-only due to evaluation time limits). The per-algorithm counts differ because the three trainers spend different numbers of environment steps per evaluation, so a fixed compute budget buys a different number of full-fidelity candidates in each; Appendix M records the alignment. All conditions run with isolated storage to prevent cross-condition contamination, and every reported champion passes the audit protocol of Section 3.3 (loader verification of all five component functions plus AST comparison against the seed program).

Base SAC hyperparameters. Our base SAC uses a 2-layer MLP (256 hidden), LayerNorm + ReLU, γ=0.99, τ=0.005, batch size 256, learning rate 3×10^{-4}, automatic entropy tuning, 1M environment steps, and 5K random exploration steps. The external references (denisyarats/pytorch_sac, CleanRL SAC) are run at identical network capacity and hyperparameters.

Evaluation protocol. All results use L2: train with seeds $\{0, 42, 123\}$, then run deterministic evaluation episodes. For SAC and PPO this is 1M training steps and 20 evaluation episodes; for DQN, whose classic-control tasks have far shorter horizons, L2 is 500K steps and 30 evaluation episodes. Baselines and evolved programs of a given algorithm always use that algorithm's L2 setting, so every comparison in this paper is within-protocol. We report mean return $\pm$ std across seeds. To guard against silent failures in the evolution pipeline, every reported champion program was additionally *audited*: we re-load the program with the component loader, verify that all five component functions are extractable, and confirm the reported fitness reproduces at L2. Only audited programs are reported.

Provenance of the reported numbers. In Table 1, the DMControl and HalfCheetah entries of the SAC blocks are independent 3-seed re-evaluations, and the Hopper/Walker2d/Humanoid entries are Aug 2026 re-evaluations. For the PPO and DQN blocks of the same table, the evolution pipeline persisted only a scalar L2 fitness for the champions, so the evolved entries are *independent* re-evaluations of the champion programs under exactly the protocol their baselines use. Because each champion was selected as the maximum over N candidates, its recorded selection score is upward-biased; the re-evaluations therefore land slightly below it on 7 of 9 tasks (by 0.001–0.071 fitness) and above it on the other two. We report the re-evaluation rather than the selection score throughout.

Why MountainCar-v0 uses 10 seeds. Its outcome is bimodal—a run either solves the task at ≈ -130 or fails at exactly -200—so a 3-seed mean is unstable: an earlier 3-seed draw caught two failures and reported -178.6 for the same champion that scores -144.9 over 10 seeds. Both sides of that comparison use 10 seeds. Over those 10, the baseline returns exactly -200.0 on every seed (std $= 0$) and the evolved program reaches the goal on 8, which is the 0/10 $\rightarrow$ 8/10 success-rate shift quoted in Section 4.1.

E Full Per-Task Results

SOTA-init effect sizes relative to seed noise. Section 4.2 reports that six of the nine SOTA-init gains move by less than a single combined seed spread (root-sum-square of the two cells' standard deviations). The per-task multiples are: Walker2d 0.04×, HalfCheetah 0.24×, Hopper 0.58×, quadruped-walk 0.71×, Humanoid 0.94×, cheetah-run 0.99×, Ant 1.34×, walker-walk 1.38×, dm-walker-run 2.22×. Each is computed from the baseline and evolved cells of the SOTA-init block below.

The full per-task table appears as Table 1 in the main body; it is not duplicated here. Its caption there carries the protocol, the marker definitions (†, §, ¶) and the per-cell provenance.

Per-task SOTA-init gaps and the head-to-head outcome. The 9-task mean gap between the two seed implementations falls from 23.4% to 13.1% in return (Section 4.2). Measured against the SOTA seed's own return, the per-task gap goes: walker-walk 29.8% $\rightarrow$ 0.2%, walker-run 35.2% $\rightarrow$ 24.2%, cheetah-run 57.6% $\rightarrow$ 24.4%, quadruped-walk 0.7% $\rightarrow$ 0.1%, HalfCheetah 53.7% $\rightarrow$ 18.2%, Hopper 4.8% $\rightarrow$ 1.8%, Walker2d 16.9% $\rightarrow$ 15.8%, Humanoid 3.2% $\rightarrow$ 6.4%, Ant 8.4% $\rightarrow$ 27.0%.

Table 5: The nine locomotion tasks both SAC batches were evolved on with 3 seeds. *Seed gap*: the SOTA-init baseline relative to the own-init baseline. *own gain* / *SOTA gain*: each batch's L2 champion relative to its own seed, i.e. what that batch's search contributed. *Final gap* and *outcome*: the two champions against each other, with the SOTA-init champion as denominator throughout. All figures are on raw returns, since fitness is normalised per task and the batches share a normalisation only within a task; a positive value always means the SOTA-init side is ahead. The final-gap column disaggregates the $23.4\% \rightarrow 13.1\%$ mean narrowing; the mean of its absolute values is 13.1%. Note that "SOTA-init" names the batch, not a claim that its seed is stronger on every task—it starts behind on quadruped-walk and Hopper.

Task	Seed gap	own gain	SOTA gain	Final gap	Outcome
dm-cheetah-run	$+57.6\%$	$+114.0\%$	$+20.0\%$	$+24.4\%$	SOTA-init
dm-walker-walk	$+29.8\%$	$+42.9\%$	$+0.5\%$	$+0.2\%$	tie
dm-walker-run	$+35.2\%$	$+20.1\%$	$+2.6\%$	$+24.2\%$	SOTA-init
dm-quadruped-walk	-0.7%	$+0.9\%$	$+1.7\%$	$+0.1\%$	tie
gym-HalfCheetah-v5	$+53.7\%$	$+77.7\%$	$+0.6\%$	$+18.2\%$	SOTA-init
gym-Hopper-v5	-4.8%	$+12.4\%$	$+20.0\%$	$+1.8\%$	SOTA-init
gym-Walker2d-v5	$+16.9\%$	$+39.8\%$	$+0.4\%$	-15.8%	own-init
gym-Humanoid-v5	$+3.2\%$	$+5.6\%$	$+9.3\%$	$+6.4\%$	SOTA-init
gym-Ant-v5	$+8.4\%$	$+1.4\%$	$+27.3\%$	$+27.0\%$	SOTA-init
Mean $\lvert$*seed gap*$\rvert$ 23.4%, *mean* $\lvert$*final gap*$\rvert$ 13.1%; *mean own gain* $+35.0\%$, *mean SOTA gain* $+9.2\%$.					

These are mean absolute deviations with the SOTA initiated seed as the denominator, and the mean of the first entry in each pair is the 23.4% quoted above; the second entries mean to 13.1%. The gap narrows on 7 of 9. It widens only on Humanoid and Ant, the two cells whose *starting* gap is among the smallest (3.2% and 8.4%), so there is little room to close and the sign is within the run-to-run spread of those cells.

Table 5 puts the two quantities side by side: how far apart the seeds were, and what each batch's own search subsequently did. Columns *own gain* and *SOTA gain* are each batch's improvement relative to *its own* seed, so they are the mechanism behind the *outcome* column rather than a second measurement of the gap.

Three readings follow. First, narrowing and outcome are different questions, and here they come apart: the gap narrows on 7 of 9 tasks, but the batch that starts ahead finishes ahead on only 4 of those 7. Quadruped-walk and Hopper both cross over—own-init starts marginally ahead and finishes marginally behind—though at $0.7\% \rightarrow 0.1\%$ and $4.8\% \rightarrow 1.8\%$ both crossings stay within a few points of zero. What separates the cases is the size of the starting advantage: all four preserved orderings sit on seed gaps of 29.8% or more, and every task where the seed gap was under 17% changed hands. A large seed advantage is durable; a small one is not.

Second, the only ordering change of any magnitude is Walker2d, and the *gains* explain it rather than the seeds. There the own-init search contributes $+39.8\%$ against the SOTA-init search's $+0.4\%$, so the batch starting 16.9% behind overtakes and finishes 15.8% ahead. Ant is the mirror case, with the SOTA-init search contributing $+27.3\%$ against $+1.4\%$—the same cell identified in Section 4.2 as the one where own-init search failed, which is why its gap widens rather than narrows. In both, the outcome follows the per-task gain, not the per-task seed.

Third, the gains are far more uneven than the seed gaps: they span $+0.4\%$ to $+114.0\%$, while the seed gaps matter only up to about 58% and, below 17%, not at all. Mean own-init gain is $+35.0\%$ against $+9.2\%$ for SOTA-init—expected, since a stronger seed leaves less headroom—but the per-task spread of that second column (from $+0.4\%$ to $+27.3\%$) is what actually decides which batch finishes ahead on the close tasks.

F PER-FAMILY QUALIFICATIONS

Cross-family comparisons are raw-return comparisons only. Fitness is normalised by task-specific bounds, and the bounds are shared across families, but the *scale* of the return differs by orders of magnitude between DQN's classic-control tasks (CartPole ceiling 500) and the locomotion tasks (HalfCheetah in the 10^4 range). A per-family median gain is therefore a statement about each family's own task suite, not a ranking of the families' algorithms; the three medians ($+39.8\%$, $+83.6\%$, $+32.1\%$) are not comparable to each other in magnitude, and none is comparable to published numbers, which use different evaluation protocols. The one cross-family comparison we do make—evolved PPO against our evolved SAC on gym-Hopper-v5—is made on raw return under one protocol (Section 4.1), which is why it is stated that way and not as a fitness difference.

The marker definitions. Three markers appear in Table 1. † marks an own-init SAC cell whose evolved return exceeds the external open-source reference for that task (denisyarats/pytorch_sac on DMControl, CleanRL SAC on Gymnasium) as we re-evaluated it at identical capacity and hyperparameters. § (on the group headers) flags that the evolved entries of that block are *independent* re-evaluations of the champion programs rather than their selection scores, which are upward-biased by max-over-N selection; the same applies to the two SAC blocks, where the re-evaluation provenance is recorded per cell in Appendix D. ¶ marks MountainCar-v0, evaluated with 10 seeds on both sides rather than 3; the reason and the full draw are in Appendix K.

The † comparisons. Two qualifications attach to a †. First, the reference numbers are our re-evaluations, not the values published by those projects, so a † means we beat our own reproduction of the reference under our protocol, not that we beat a leaderboard entry. Second, all five † cells sit in the own-init block, where the comparison crosses a change of seed program as well as a change of scale—five of the nine locomotion tasks are cleared from our own weaker seed (walker-walk, quadruped-walk, Hopper, Walker2d, Humanoid), and the SOTA-init block re-runs the same nine to isolate the scale effect, where the margins are much smaller (median $+2.6\%$).

The MountainCar normalisation artifact. MountainCar-v0's baseline scores a fitness of 0.908 while returning exactly -200.0 on every seed, i.e. the worst attainable return for that environment. The task's normalisation bounds place -200 near the top of the fitness range, so the worst possible policy receives a high fitness. This is why the DQN ablation reports all conditions within 0.908–0.948 on that task (Section 5) and why we read no ordering from that column: it compresses the entire failure region into a narrow high-fitness band, and the task cannot separate conditions there. The relevant MountainCar result is the success-rate shift under 10 seeds ($0/10 \rightarrow 8/10$), not any fitness gap, and we report it that way.

G BENCHMARK TASK DETAILS

Table 6: SAC benchmark tasks with observation/action dimensions and normalization bounds. All 17 are evolved in the own-init batch; the 9 locomotion tasks (all DMControl and all Gymnasium entries) additionally have an external SOTA reference and were also run in the SOTA-init batch, all 9 of which are reported in Table 1. The cross-algorithm experiments additionally use gym-Hopper/HalfCheetah/Ant/Walker2d/Humanoid-v5 for PPO and CartPole-v1, MountainCar-v0, Acrobot-v1, LunarLander-v3 for DQN.

Task	Source	Obs Dim	Act Dim	Norm Bound
dm-cheetah-run	DMControl	17	6	1000
dm-walker-walk	DMControl	24	6	1000
dm-walker-run	DMControl	24	6	1000
dm-quadruped-walk	DMControl	78	12	1000
gym-HalfCheetah-v5	Gymnasium	17	6	15000
gym-Hopper-v5	Gymnasium	11	3	4000
gym-Walker2d-v5	Gymnasium	17	6	6000
gym-Ant-v5	Gymnasium	105	8	8000
gym-Humanoid-v5	Gymnasium	348	17	7000
mw-push-v3	Meta-World	39	4	4500
mw-drawer-open-v3	Meta-World	39	4	4500
mw-door-open-v3	Meta-World	39	4	4500
mw-peg-insert-side-v3	Meta-World	39	4	4500
mw-pick-place-v3	Meta-World	39	4	4500
md-validation	MetaDrive	259	2	350
md-10env	MetaDrive	259	2	350
md-100envs	MetaDrive	259	2	350

H Cross-Task Generalization

A critical question is whether evolved programs are overfit to their training task or discover generalizable improvements. Table 7 gives the full transfer matrix in numeric form: 4 champion programs (evolved on dm-walker-walk, dm-quadruped-walk, gym-Walker2d-v5, and gym-Humanoid-v5) re-evaluated under the same 3-seed L2 protocol as Table 1 on 9 locomotion targets spanning DMControl and Gymnasium, alongside each target's fixed baseline fitness so that every Δ can be read as a fraction of its baseline. Figure 2b in the main body renders the same 36 cells as a heatmap.

Counting only non-source cells, the **Humanoid program** generalizes best: 7/8 targets improved at mean $+0.157$ fitness, and it is the best source program on 4 of the 9 rows. The **walker-walk program** ties it on win count (7/8) at a smaller mean ($+0.097$) despite being evolved on the simplest task in the suite, and is likewise best on 4 rows; its single failure is gym-Hopper-v5 (-0.207). The **quadruped program** improves 6/8 ($+0.083$); its modifications (symlog encoding, cosine-annealed Q blending, Log-Cosh critic loss) are broadly beneficial but lose on gym-Hopper-v5 and gym-Ant-v5. The **walker2d program overfits**: 2/8 wins at mean -0.134, including -0.669 on dm-quadruped-walk, relying on step-dependent strategies with hard-coded thresholds. Two caveats bound how far this ordering can be pushed. Each column is a single program, so differences between columns conflate the source task with the particular champion that task happened to produce; and the walker2d column describes a program that later evolution superseded (see caption), so its poor transfer is not established for the champion of Table 1. The principles behind the difference are distilled below.

What makes programs generalizable? Comparing a well-generalizing program (quadruped, 6/8 wins) with the worst (walker2d, 2/8 wins), we identify three principles for transferable RL modifications. **(1) Relative scheduling**: use progress ratio t/T rather than absolute step counts for annealing—the quadruped program's cosine-annealed schedules adapt to any training length. **(2) Stateless encoding**: pointwise transformations (arcsinh, symlog) transfer perfectly; running statistics do not. **(3) Adaptive thresholds**: parameters derived from training dynamics (EMA of TD

Table 7: Cross-task L2 generalization, full 9×4 matrix: fitness improvement (Δ) over the fixed L2 baseline when a champion evolved on one task is re-evaluated on each target. Source task marked with *; the summary rows exclude source cells. Bold = best source program for that target. *Provenance.* The matrix was evaluated as a single batch using the champions available at that time. The gym-Walker2d-v5 own-init champion was subsequently improved by resumed evolution (fitness $0.760 \rightarrow 0.905$), so the entire walker2d column describes the *earlier* program. We did not substitute the newer champion into individual cells: only its own-task value was re-measured, so patching the diagonal alone would leave a column mixing two different programs and therefore not internally comparable. Two consequences follow. The walker2d diagonal here ($0.647 + 0.113 = 0.760$) is below the gym-Walker2d-v5 own-init row of Table 1 (0.905); and the weak transfer of that column is a property of the earlier program, *not* an established property of the champion reported in Table 1.

	Source program (Δ fitness)				
Task	walker-walk*	quadruped*	walker2d*	Humanoid*	Baseline Fitness
dm-cheetah-run	+0.265	+0.269	−0.129	**+0.313**	0.302
dm-walker-walk	**+0.292***	+0.290	+0.243	+0.221	0.680
dm-walker-run	+0.091	+0.025	−0.319	**+0.142**	0.488
dm-quadruped-walk	**+0.019**	+0.006*	−0.669	−0.082	0.932
gym-HalfCheetah-v5	+0.058	**+0.154**	−0.148	+0.141	0.340
gym-Hopper-v5	−0.207	−0.076	−0.001	**+0.201**	0.619
gym-Walker2d-v5	+0.028	+0.003	+0.113*	**+0.163**	0.647
gym-Ant-v5	**+0.239**	−0.056	+0.188	+0.155	0.467
gym-Humanoid-v5	**+0.279**	+0.052	−0.234	+0.056*	0.714
Cross-task wins	7/8	6/8	2/8	**7/8**	—
Mean Δ (excl. source)	+0.097	+0.083	−0.134	**+0.157**	—

errors, as in the walker-walk program) auto-calibrate to each task, while hard-coded constants may be inappropriate for different reward scales.

I Discovered Algorithmic Innovations: Detailed Analysis

Huber loss addresses bootstrapping noise. SAC's critic is trained on bootstrapped targets that are inherently noisy early in training. MSE amplifies large errors quadratically, destabilizing learning. Huber loss caps the gradient at δ, providing natural gradient clipping without explicit hyperparameter tuning.

Q-annealing balances exploration-exploitation. The pessimism of $\min(Q_1, Q_2)$ underestimates state values, which can suppress exploration. Early optimism (toward mean) encourages visiting novel states; progressive conservatism later prevents overestimation from corrupting the policy.

Symlog prevents saturation. In tasks like MetaDrive (259D LiDAR observations) or HalfCheetah (high-velocity states), raw observations can reach extreme values that saturate ReLU networks. Logarithmic compression keeps inputs in a network-friendly range.

Robust critic loss (16/17 tasks; EMA-adaptive scale in 9). The most universal discovery replaces MSE with a robust loss (Huber, smooth-L1, Log-Cosh, or Charbonnier); 9 champions further adapt the transition scale from running error statistics—8 as a Huber δ, and md-10env as a Log-Cosh κ clamped to $[1, 5]$:

Listing 1: Evolved adaptive Huber loss (dm-walker-walk, simplified)

```
# Track TD-error distribution with EMA
td_errors = (q_pred - q_target).abs()
self.ema_delta = 0.99 * self.ema_delta + 0.01 * td_errors.quantile(0.9)

# Asymmetric weighting based on Q-disagreement
q_disagreement = (q1 - q2).abs().mean()
tau = 0.55 + 0.2 * sigmoid(q_disagreement - self.ema_disagreement)

# Asymmetric Huber: overestimation penalized more heavily
residual = q_pred - q_target
loss = torch.where(
    residual.abs() < self.ema_delta,
    0.5 * residual**2 / self.ema_delta,
    residual.abs() - 0.5 * self.ema_delta
)
loss = torch.where(residual > 0, tau * loss, (1-tau) * loss)
```

Progressive Q-blending (12/17 tasks). Standard SAC uses $\min(Q_1, Q_2)$ (pessimistic). Evolution discovers progress-annealed blending of $\min$ and mean. A representative locomotion schedule (dm-walker-walk): early training ($t/T < 0.3$) uses $\text{mean}(Q_1, Q_2)$ for optimistic exploration, shifting via sinusoidal annealing to pure $\min$ for conservative exploitation. Manipulation and driving champions typically anneal in the opposite direction, relaxing pessimism as training stabilizes.

Symlog encoding (11/17 tasks). Evolution discovers $\text{encode}(x) = \text{sign}(x) \cdot \log(1 + |x|)$ or equivalently $\text{arcsinh}(x/\sigma) \cdot \sigma$, preserving sign while compressing dynamic range. This was independently proposed in DreamerV3 (Hafner et al., 2023).

J PPE Efficiency

Table 8 quantifies PPE's filtering across the 6 locomotion tasks. Each run evaluates 230 L0 candidates; evaluating all of them directly at L2 (∼5.8 GPU-hours each) would require ∼1330 GPU-hours per run. With PPE, L0 screening costs ∼37 GPU-hours, the surviving ∼18 candidates cost ∼21 GPU-hours at L1, and the final 5 candidates cost ∼29 GPU-hours at L2—roughly 87 GPU-hours in total, a **15× reduction**. The L0→L1 promotion rate of ≈7.6% means PPE eliminates 92% of candidates at minimal cost.

Table 8: PPE filtering efficiency across 6 locomotion tasks. Statistics are per run, over the full B=240 candidate budget.

Metric	Mean	Min	Max
L0 evaluations per run	230	230	230
L0→L1 promotion rate	7.6%	4.8%	12.2%
Code deduplication rate	18%	2.5%	30%
Fraction of generated candidate programs discarded as AST-duplicates before screening; the denominator is the B=240 budget, so ≈230 survive to L0.			
L1 evaluations per run	17.5	11	28
L2 evaluations per run	5	5	5
GPU-hours per run (L0 / L1 / L2)	36.8 / 21.1 / 28.9	—	—

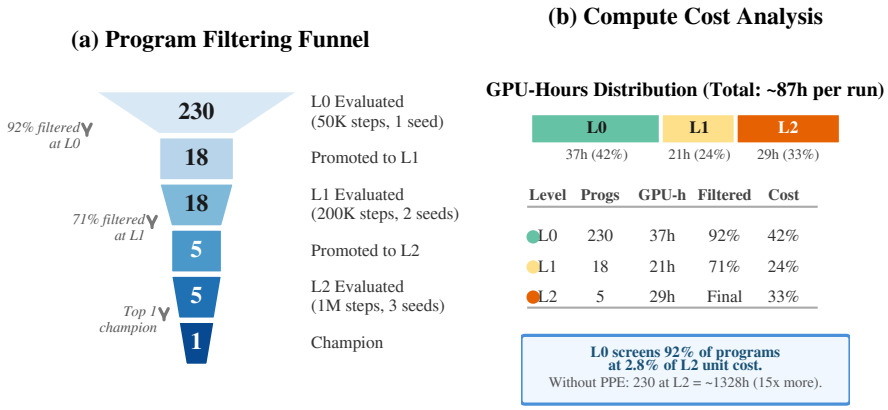


Figure 3: PPE filtering efficiency (per-run averages). **(a)** Program filtering funnel: from 230 L0-evaluated candidates to 5 L2-validated finalists. L0 filters 92% of programs at minimal cost. **(b)** GPU compute distribution: total ∼87 GPU-hours per run versus ∼1330 for evaluating all candidates at L2, a ∼15× reduction.

Cross-level rank agreement. The filtering claim of Section 2 rests on the assumption that low-fidelity returns rank candidates in roughly the order their L2 returns would. We test that directly rather than assuming it. Pooled over the ablation runs (A0, A1 and w/o Phase 1; n=466 candidate triplets) the Spearman correlation with L2 fitness is $+0.80$ for L1 ($p \approx 2\times10^{-105}$, block-bootstrap 95% CI $[+0.72, +0.86]$, resampling whole runs) against $+0.73$ for L0, and the L1 figure is stable across all three conditions ($+0.77$/$+0.79$/$+0.83$). L1 is therefore a sound proxy for L2, which is what justifies spending L1 budget on the ∼18 survivors rather than promoting straight from L0. The L0 figure is weaker evidence: much of its pooled $+0.73$ comes from cross-task scale differences rather than from within-run ordering, and the correct within-run reading is confined to single-layer conditions. We report the pooled correlations as a check on the funnel's ordering, not as a claim that L0 ranking is reliable.

Correlation with L2 fitness	ρ	n	95% CI
L1 → L2 (A0 + A1 + w/o Phase 1)	$+0.801$	466	$[+0.72, +0.86]$
L0 → L2 (same pool)	$+0.727$	466	—
L1 → L2, A0 only	$+0.774$	160	—
L1 → L2, A1 only	$+0.790$	160	—
L1 → L2, w/o Phase 1 only	$+0.830$	146	—

K Full Per-Seed Results

Table 9 gives the per-seed L2 returns behind every mean ± std in Table 1, for all 17 SAC own-init tasks plus the 5 PPO and 4 DQN tasks. Seeds are the training seeds $\{0, 42, 123\}$ under each algo-

rithm's own L2 protocol (SAC/PPO 1M steps and 20 episodes; DQN 500K steps and 30 episodes); the $\pm$ in Table 1 is the population standard deviation of these three numbers. The spread is highly task-dependent, and reading the individual seeds changes how several headline rows should be interpreted. Within SAC, the two Meta-World tasks with order-of-magnitude gains are also the two least reliable: mw-pick-place-v3 solves the task on one seed out of three (2887.3 vs. 2.9 and 3.9) and mw-peg-insert-side-v3 on two out of three; those rows should be read as "evolution finds a solution that sometimes trains" rather than as a stable improvement. In DQN, the evolved CartPole program hits the environment's 500-return ceiling on two of three seeds, so its ± 69.0 is one-sided censoring rather than symmetric noise, and the LunarLander baseline spans -31.6 to 180.0 (it crashes on the first seed) while all three evolved seeds land—which is the substance of that row's improvement, more than the change in mean. PPO's per-seed spreads are otherwise unremarkable, with gym-Hopper-v5 the tightest cell in the whole table (3575.1 / 3444.4 / 3556.9).

Table 9: Per-seed L2 returns for all 17 SAC own-init tasks and all 9 PPO/DQN tasks (training seeds $\{0, 42, 123\}$; SAC/PPO 1M steps and 20 evaluation episodes, DQN 500K steps and 30 episodes). These are the values summarized in Table 1. ¶ MountainCar-v0 is the one exception: because its outcome is bimodal (solve at ≈ -130 or fail at -200), Table 1 reports a 10-seed re-evaluation rather than a 3-seed one. The three seeds shown here are the $\{0, 42, 123\}$ subset of that draw; the full 10 evolved returns are -120.7, -141.2, -200.0, -153.9, -124.8, -200.0, -128.8, -124.4, -134.7, -120.6 (mean -144.9), and the baseline returns exactly -200.0 on all 10. The 3-seed subset therefore does *not* average to the reported mean.

	Baseline			Evolved		
Task	Seed 0	Seed 42	Seed 123	Seed 0	Seed 42	Seed 123
SAC — DMControl						
dm-cheetah-run	344.0	309.4	253.5	612.4	741.1	587.3
dm-walker-walk	626.0	687.4	725.5	974.3	972.4	966.5
dm-walker-run	553.1	415.6	494.4	583.7	595.2	578.1
dm-quadruped-walk	936.2	914.5	944.0	938.2	935.8	945.4
SAC — Gymnasium						
gym-HalfCheetah-v5	5255.9	4798.0	5226.8	11164.8	7468.8	8527.6
gym-Hopper-v5	2441.5	1740.7	3242.8	3256.9	1571.4	3514.0
gym-Walker2d-v5	4044.3	4030.5	3574.6	5797.4	4743.7	5747.3
gym-Humanoid-v5	5261.6	4974.8	4760.0	5484.6	5116.7	5242.3
gym-Ant-v5	4458.2	3439.8	3300.5	3129.7	3283.1	4944.8
SAC — Meta-World						
mw-push-v3	5.6	4.7	4.8	240.2	4459.3	776.0
mw-drawer-open-v3	1493.1	3988.8	4049.1	4562.4	4607.1	4694.5
mw-door-open-v3	3621.1	3445.1	1651.8	380.6	4481.8	4437.4
mw-peg-insert-side-v3	25.7	5.3	3424.7	4607.0	2.7	4617.0
mw-pick-place-v3	6.8	9.8	3.9	2.9	2887.3	3.9
SAC — MetaDrive						
md-validation	266.1	112.5	65.0	459.9	395.8	114.0
md-10env	304.2	164.0	220.7	223.2	205.3	269.5
md-100envs	311.7	164.4	235.1	254.4	282.9	309.2
PPO — Gymnasium						
gym-Hopper-v5	2128.4	1510.0	2121.6	3575.1	3444.4	3556.9
gym-HalfCheetah-v5	1682.9	1712.5	1777.6	3837.5	4175.1	4688.3
gym-Ant-v5	795.4	858.1	674.4	2411.5	2541.7	3413.9
gym-Walker2d-v5	4229.1	3138.4	4702.6	4357.8	3899.9	5062.8
gym-Humanoid-v5	561.7	573.9	684.2	938.8	744.4	744.0
DQN — classic control						
CartPole-v1	311.9	354.8	324.0	500.0	500.0	353.6
MountainCar-v0¶	−200.0	−200.0	−200.0	−120.7	−141.2	−200.0
LunarLander-v3	−31.6	52.2	180.0	150.6	165.4	207.8
Acrobot-v1	−92.8	−84.1	−96.8	−80.1	−84.9	−95.9

L Evaluation Wall-Clock Caps and Their Effect on the Ablation

Every evaluation runs under a per-level wall-clock cap, and every run under a total cap. Because the ablation conditions differ in how expensive a single evaluation is, these caps do not bind equally, and Section A summarises the consequence. The per-condition figures are as follows.

(1) L0 cap (1200 s). It truncates 40–65% of PPO L0 evaluations. All L0-screening conditions are affected equally, so there is no condition-level selection bias, but the L0 signal is noisier than it would otherwise be.

(2) Headroom differs by condition. The L2-only cap (43200 s) is $7.3\times$ its median evaluation cost and is therefore structurally immune to truncation, whereas L0-only ($1.95\times$ headroom) and L1-only ($1.76\times$) each lose roughly half their samples under slow-tail conditions. Part of L2-only's apparent advantage in Table 10 therefore has a structural rather than methodological origin.

(3) SAC L1-only is right-censored. It is the only condition for which the 4800 s L1 cap binds, so its Table 10 entry is a right-censored estimate of what an uncapped L1-only run would reach.

(4) Run-level cap (96 h). This truncated 4 of the 6 PPO L2-only runs across both PPO pairs (35/42/34/36 of 43 planned L2 evaluations); all DQN L2-only runs reached their full 33. Truncation removes candidates from L2-only and thus works *against* it, so the reported PPO L2-only margin over the full method is a conservative lower bound.

M Full Component-Ablation Matrix

Table 2 summarises the component ablation; Table 10 gives every cell. For completeness we also report the exact two-sided Wilcoxon signed-rank test of each ablation against the full method over the 8 paired differences, in the column order of Table 10: $p = 0.383/0.250/0.195/0.383/0.312$. The smallest value attainable at $n{=}8$ is 0.008, so this design has little power against effects of the observed size; we report the paired $\Delta\pm$ std in the main text for that reason. Mean ranks over the six conditions within each pair (average-rank convention for ties) are $3.06/3.50/3.69/4.38/2.88/3.50$ over all 8 pairs and $2.50/3.25/3.75/4.33/3.50/3.67$ over the 6 non-degenerate pairs: the full method is second to L2-only on the first grouping and first on the second. Two cells are degenerate and marked *: PPO on gym-HalfCheetah-v5, whose low values reflect PPO's own weakness on that task under the shared normalisation rather than any ablation effect, and MountainCar-v0, which is saturated (Appendix O). Neither enters the SAC mean, nor the 6-pair rows of Table 2. Because the matrix is complete, the *differences* along the 8-pair mean row are identical to the mean paired Δ of each ablation against the full method; the absolute level of that row has no interpretation, since fitness is not comparable across algorithm families.

Table 10: Per-task component ablation: audited L2 champion fitness, mean over 3 independent evolution runs per cell (evolution seeds 42/123/2026). Best per row in bold.

		Ablating PCE		Ablating PPE		
Task	Full	w/o decomp.	w/o Phase 1	L0-only	L2-only	L1-only
SAC						
dm-cheetah-run	**0.685**	0.639	0.622	0.556	0.613	0.636
gym-HalfCheetah-v5	**0.763**	0.702	0.694	0.682	0.649	0.671
md-100envs	0.907	0.945	0.954	**0.957**	0.754	0.728
mw-drawer-open-v3	0.780	0.763	0.763	0.718	**0.782**	0.730
SAC mean	**0.784**	0.762	0.758	0.728	0.700	0.691
PPO						
gym-Hopper-v5	0.849	0.874	0.818	0.850	**0.882**	0.876
gym-HalfCheetah-v5*	0.201	0.201	0.197	0.193	**0.272**	0.204
DQN						
LunarLander-v3	0.752	0.465	0.638	0.600	0.689	**0.772**
MountainCar-v0*	0.908	0.909	0.948	0.942	**0.959**	0.938
Mean over all 8 pairs	**0.731**	0.687	0.704	0.687	0.700	0.694

N PER-CELL EVOLUTION VARIANCE OF THE SAC ABLATION

Table 11 restates the SAC block of Table 10 with the per-cell standard deviation across the three independent evolution runs, together with the unevolved SAC baseline under the identical L2 protocol. These spreads set the resolution of the PCE comparison: the gaps being compared (0.02–0.04) are smaller than several of the cell standard deviations.

Table 11: SAC component ablation with per-cell evolution variance: audited L2 champion fitness, mean ± std over **3 independent evolution runs** (evolution seeds 42/123/2026). The ± here is *evolution* variance, not the *training* variance reported in Table 1; the two are different quantities and should not be compared in magnitude. ‡ The baseline was never evolved, so no evolution variance exists for it; the spread on that row is its *training* variance over the three seeds $\{0, 42, 123\}$ and is not comparable to the evolution variance above it. Best per column in bold.

Condition	dm-cheetah-run	HalfCheetah-v5	md-100envs	mw-drawer-open	Avg
Full method	**0.685** ± 0.025	**0.763** ± 0.048	0.907 ± 0.095	0.780 ± 0.027	**0.784**
Ablating PCE					
w/o decomposition	0.639 ± 0.001	0.702 ± 0.083	0.945 ± 0.017	0.763 ± 0.051	0.762
w/o Phase 1	0.622 ± 0.018	0.694 ± 0.109	0.954 ± 0.011	0.763 ± 0.029	0.758
Ablating PPE					
L0-only selection	0.556 ± 0.059	0.682 ± 0.071	**0.957** ± 0.044	0.718 ± 0.035	0.728
L2-only selection	0.613 ± 0.007	0.649 ± 0.012	0.754 ± 0.136	**0.782** ± 0.024	0.700
L1-only selection	0.636 ± 0.019	0.671 ± 0.022	0.728 ± 0.150	0.730 ± 0.156	0.691
SAC baseline‡	0.302 ± 0.037	0.340 ± 0.014	0.677 ± 0.172	0.706 ± 0.265	0.506

O DQN COMPONENT ABLATION

We repeated the component ablation of Section 5 (full method vs. w/o Phase 1 vs. w/o component decomposition) on the two DQN tasks, 3 independent evolution runs per cell.

Only LunarLander is informative. MountainCar is saturated: all three conditions land within 0.908–0.948 (and within 0.91–0.96 across all six conditions once the PPE ablations are included), a spread comparable to the within-cell seed noise, so the task cannot separate the conditions—and there the no-Phase-1 condition is nominally *ahead* of the full method. We report the MountainCar column for completeness but draw no conclusion from it, and we do not average the two tasks together.

On LunarLander the SAC ordering holds directionally—full 0.752 > w/o Phase 1 0.638 > w/o decomposition 0.465, a +0.287 margin over the monolithic condition. With n=3 runs per condition, however, the smallest attainable one-sided Wilcoxon p is 0.125, so significance is out of reach by construction; this is directional replication only.

Table 12: DQN component ablation (3 independent evolution runs per cell; mean ± std of the audited L2 champion fitness). LunarLander replicates the SAC ordering full > w/o Phase 1 > w/o decomposition directionally. MountainCar is saturated—all conditions within 0.908–0.948—and is not read; no mean is taken across the two tasks. Best per row in bold.

Task	Full	w/o Phase 1	w/o decomp.
LunarLander-v3	**0.752** ± 0.027	0.638 ± 0.064	0.465 ± 0.084
MountainCar-v0 (saturated)	0.908 ± 0.006	**0.948** ± 0.009	0.909 ± 0.049

P COMPONENT-LEVEL ANALYSIS

Tables 13 and 14 describe the five component implementations of all 17 champion programs of Table 1. Each cell was written by reading the corresponding function in the champion's source file, not by paraphrasing the LLM's own commit message; descriptions name the mechanism that

is actually executed on the SAC code path. All 17 programs additionally carry an unreachable on-policy (GAE/PPO) branch inherited from the shared function template; those branches are ignored here because they never execute under SAC.

Table 13: Component implementations of the champion programs, DeepMind Control and Gym locomotion (9 tasks). "Standard SAC" and "unchanged" mark slots that are *not* meaningfully evolved away from the seed program; p denotes training progress in $[0, 1]$.

Task	Encoding	Value Target	Critic Loss	Policy Loss	Exploration
DeepMind Control					
dm-cheetah-run	$\operatorname{arcsinh}(x)$, no rescale	Fixed $0.95\min + 0.05\max$	Huber (β=1) $+ 10^{-5}Q^2$	Standard SAC $+ 10^{-3}$ mean L2	$5\times$ reward, EMA curiosity, action L1
dm-walker-walk	$5\operatorname{arcsinh}(x/5)$, clip $[-25, 25]$	Sine anneal mean→min, then variance penalty	Asym. Huber, EMA-adaptive δ, τ 0.55–0.75	Cosine min/max blend; asym. α LR $5\times/1\times$	Dual-timescale EMA novelty, cosine decay
dm-walker-run	$5\operatorname{arcsinh}(x/5)$ + decaying noise	Per-sample min/mean from Q-disagreement	Asym. Huber, quantile-EMA δ	Softmin Q (β 2→10), PI α controller	Convexified reward + movement synergy
dm-quadruped-walk	Clip $[-20, 20]$, then symlog	Cosine-annealed max-Q weight $0.1 \to 0$	Scaled log-cosh (β=2) $+ 10^{-5}Q^2$	Standard SAC $+ 10^{-3}$ action L2	$10\times$ reward + decaying movement bonus
Gym MuJoCo					
gym-HalfCheetah-v5	Clip $[-20, 20]$ only	Min-weight anneal $0.5 \to 0.95$	Standard Huber (β=1)	Standard SAC $+ 10^{-3}$ mean L2	Welford z-score novelty, off at p=0.6
gym-Hopper-v5	Piecewise log above $\lvert x\rvert$>5, clip $[-20, 20]$	**Standard SAC** $\min(Q_1, Q_2)$	Huber, fixed δ=10	**Standard SAC**	**None**—returns reward unchanged
gym-Walker2d-v5	Symlog (scale 5) + noise over first 10%	Softmin Q, temperature $1.0 \to 0.1$	Asym. Charbonnier, τ $0.40 \to 0.60$	Min→mean blend + disagreement penalty	RFF-RND novelty, e^{-4p} decay, action L2
gym-Humanoid-v5	Symlog + deadzone, clip $[-20, 20]$	Optimistic min/max blend, weight $\to 0$	Huber, δ=EMA|TD|, growing asymmetry	**Standard SAC**	Random-projection Mahalanobis novelty
gym-Ant-v5	Clip $[-20, 20]$ only	**Standard SAC** $\min(Q_1, Q_2)$	**Plain MSE (unchanged)**	Standard SAC $+ 10^{-3}$ mean L2	$0.05\log$ of EMA-scaled $\lVert\Delta\text{obs}\rVert$, linear decay

Table 14: Component implementations of the champion programs, Meta-World manipulation and MetaDrive driving (8 tasks). On these benchmarks the encoder is where most of the evolved content sits: every champion rewrites the raw observation vector into task-specific geometric features.

Task	Encoding	Value Target	Critic Loss	Policy Loss	Exploration
Meta-World					
mw-push-v3	Unused dims → distance/velocity features, clip $[-10, 10]$	Softmin blend, κ 5.0 → 0.5, $w \le 0.4$	Asym. Huber, EMA δ, τ $0.8 \to 0.5$	Batch-normalized Q + UCB disagreement bonus	Potential shaping (c=50/200) + EMA curiosity
mw-drawer-open-v3	Inject distance/contact/alignment scalars, then symlog	Variance-aware min/mean, pessimism $1.0 \to 0.2$	Quantile-EMA Huber (0.85), TD reweighting	Min-Q − decaying disagreement; action penalties	Adaptive potential shaping, decay floor 0.2
mw-door-open-v3	Velocity rebuild + proximity features, scaled symlog	Min-Q $+ 0.25$ cosine-decayed optimism	Log-cosh Huber, EMA δ, $+ 10^{-4}Q^2$	$0.95\min + 0.05\max$; 0.05 disagreement penalty	Potential shaping + object motion + grasp term
mw-peg-insert-side-v3	History dims → velocity/direction features	Min-Q $+ \beta(\max - \min)$, β $0.15 \to 0$	Standard Huber, MAD-derived EMA δ	**Standard SAC** (+ inactive saturation hinge)	Potential shaping (inverse/exp/grasp), $0.30 \to 0.05$
mw-pick-place-v3	Overwrite dims 18–29 with TCP/object/goal vectors	Min-Q + ≤ 15% disagreement-gated optimism	Huber, δ=EMA 80th-pct TD error	Standard SAC + mean L2 annealed to 0	Potential shaping on reach/place/z distances
MetaDrive					
md-validation	Symlog ego; LiDAR → forward-weighted sector statistics	$0.9\min + 0.1\max$, target clamp $[-50, 2000]$	Expectile-weighted Huber (β=2)	Boundary penalty + per-channel action regularizers	Symlog reward + EMA ego novelty, off at p=0.5
md-10env	EMA z-score ego + tanh clip; LiDAR 3-beam max-pool	Mean − trust-gated disagreement; γ $0.90 \to 1.00$	Asym. log-cosh, EMA $\kappa \in [1, 5]$	Mean − pessimism · std; PI α controller	LiDAR safety margin + novelty + stagnation penalty
md-100envs	Symlog ego; LiDAR avg/min smoothing → $(1-d)^2$	Pessimism $0.5+0.5p$ blend of min and mean	Focal-weighted adaptive-δ Huber	Advantage-weighted BC trust region + boundary penalty	LiDAR approach-rate penalty + throttle curriculum

Read across all $17 \times 5 = 85$ slots, the tables support a more restrained reading of the method than the per-task highlights of Appendix I do, and we state the three caveats explicitly.

Not every component slot is meaningfully evolved. Three of gym-Ant-v5's five components are the unmodified seed program—a clip-only encoder, a textbook $\min(Q_1, Q_2)$ target, and plain MSE critic loss—and gym-Hopper-v5's exploration component consists of `return float(reward)`, i.e. the slot is switched off entirely. Four champions (Hopper, Humanoid, peg-insert-side, pick-place) retain the standard SAC actor loss up to a small action regularizer. Phase 1 spends its bandit pulls on all five arms regardless, so an evolved program is not a program in which all five components changed; it is one in which the components that mattered changed. This is consistent with the ablation result of Section 5, where the margin of per-component mutation over rewriting the whole program is smaller than the run-to-run spread of the cells being compared.

Several annealed mechanisms converge back to the baseline. dm-quadruped-walk's max-Q weight decays $0.1 \to 0$, gym-Humanoid-v5's optimism weight decays to 0, and gym-Walker2d-v5's softmin temperature decays to a hard minimum. In each case the champion is a deviation from clipped double-Q early in training and is exactly clipped double-Q at convergence.

P.1 The PPO and DQN Champions

Tables 15 and 16 give the same slot-by-slot reading for the 5 PPO and 4 DQN champions of Table 1. As above, every cell describes the mechanism that actually *executes* under that family's trainer rather than the mutation's own rationale comment, and "unchanged" is measured against the vanilla seed program (identity encoder, textbook GAE / one-step Bellman target, plain MSE, textbook clipped surrogate, unshaped reward), not against the champion's own run directory—each run's local starting program is already a Phase-1 evolved program, so diffing against it would systematically understate what evolution changed.

Which slots are live under each trainer. This determines what the counts below can mean. Our PPO trainer passes the on-policy value array and λ into the value-target component and never passes twin-Q arguments, so the off-policy branch every champion inherits from the shared template is unreachable—the mirror of the unreachable GAE branch in the SAC champions. Our DQN trainer calls only *four* of the five components: `compute_policy_loss` is never imported, and the value-target and critic-loss components are called with the twin-Q arguments, the entropy coefficient and the second critic's predictions all set to null, which makes the GAE branch, the twin-Q aggregation branch, the entropy-bonus term and the second-critic term dead in all four DQN champions. Crucially, the two mechanisms a reader would most expect a DQN champion to innovate on—the ϵ-greedy schedule and the target-network synchronisation—are hardcoded harness hyperparameters *outside all five slots*, so DQN evolution could never reach them. This is a property of our decomposition as implemented, not a discovery failure, and it is the single most important constraint on the cross-family comparison.

PPO innovates in the slots that are characteristically PPO, in every champion. All 5 PPO champions modified all five slots. More informative than the count is *where* the content sits: 5/5 value-target modifications are GAE-specific (λ scheduling in 4/5—three linear, one cosine—dual-λ streams in 1/5, Monte-Carlo blending in 1/5, advantage centering in 1/5), and no champion touched the discount or the bootstrap structure. All 5 policy-loss modifications are clipped-surrogate-specific: dual clipping 5/5 and KL regularisation 5/5, the latter in four distinct forms (fixed coefficient, hinge, adaptive multiplicative, and mask-based pseudo early-stopping), plus clip-range annealing 4/5 and ratio or advantage clamping 4/5. These mechanisms have no SAC or DQN analogue, and the SAC signature of Q-annealing has no PPO analogue because PPO has no twin critics.

DQN's live surface is three slots, and it uses all three. Counting only reachable code, the DQN champions modified encoding 4/4, exploration 4/4 and the critic loss 3/4 (asymmetric Huber on Acrobot, expectile regression on MountainCar, Huber on LunarLander; CartPole's $0.5\times$MSE is a constant rescaling absorbable into the learning rate and we score it unchanged). The live value target is the seed $\max_a Q'$ backup in 4/4, and the policy slot is 0/4 by construction. Two independent implementations of RND with random Fourier features arose on MountainCar and Acrobot, which is the clearest evidence that evolution used the room it did have. Counting dead code would have turned the value-target column from 0/4 into 4/4, so we state the live counts and not the nominal ones.

Table 15: Component implementations of the 5 PPO champion programs. p denotes training progress in $[0, 1]$. Every champion inherits an unreachable off-policy (twin-Q) branch from the shared template, which is ignored here. § Entropy term with a sign *opposite* to our trainer's convention: see the caveats below.

Task	Encoding	Value Target	Critic Loss	Policy Loss	Exploration
gym-Hopper-v5	Piecewise symlog above $\|x\|>5$, clip $[-15, 15]$	Cosine-annealed GAE λ 0.98 $\rightarrow$ 0.90 + MC-return blend (weight $0.1p$)	Smooth-L1, β 1.0 $\rightarrow$ 0.2, $+10^{-4}V^2$	Dual clipping (c=3) + multiplicatively adapted KL coef. (target 0.015) + decaying entropy§	EMA-normalized state-difference novelty $-$ action L2 + survival bonus
gym-HalfCheetah-v5	Symlog $\text{sign}(x)\log(1+\|x\|)$	Linear GAE λ 0.98 $\rightarrow$ 0.90	Huber (β=1) + decaying $5\times10^{-5}V^2$	Clip anneal 0.2 $\rightarrow$ 0.1, adv. clamp ±5, KL penalty $0.02(1-p)$, entropy	$-0.005\,\mathbb{E}[a^2]$ + decaying state-magnitude bonus
gym-Ant-v5	$5\tanh(x/5)$ + decaying Gaussian noise	Linear GAE λ 0.95 $\rightarrow$ 0.85 + advantage mean-centering	Adaptive Huber, β 2.0 $\rightarrow$ 0.5, $+10^{-4}V^2$	Dual clipping (c=1.5) + KL penalty 0.005 + entropy $0.01e^{-3p}$	Dimension-invariant action penalty + decaying survival bonus
gym-Walker2d-v5	Symmetric clip $[-10, 10]$	Linear GAE λ 0.95 $\rightarrow$ 0.90	Adaptive-δ Huber, δ= batch median $\|$TD$\|$, clamp $[0.1, 10]$	Dual clipping (c=3) + hinge KL penalty $\text{relu}(\text{KL}-0.015)$, entropy	Decaying action-energy penalty $-0.005\Sigma a^2(1-p)$
gym-Humanoid-v5	Piecewise log above $\|x\|>2$, linear below	**Dual-λ GAE**: parallel λ=0.97 and 0.90 streams, averaged 50/50	Huber, fixed β=10 (MSE-like in range, linear on outliers)	Clip anneal 0.2 $\rightarrow$ 0.05 + dual clipping + KL-mask pseudo early-stopping + KL penalty	Decaying L1 *and* L2 action penalties, clip ±10

Table 16: Component implementations of the 4 DQN champion programs. Bracketed *[dead]* annotations name mechanisms present in the file but unreachable under our DQN trainer; they are excluded from all counts. The policy-loss slot is never called at all, so no DQN champion can innovate there.

Task	Encoding	Value Target	Critic Loss	Policy Loss	Exploration
CartPole-v1	Welford online standardization $\rightarrow$ symlog $\rightarrow$ clip ±20	**Unchanged** $r+\gamma(1-d)\max_a Q'$ *[dead:* $0.8\min+0.2\max$ *aggregation]*	$0.5\times$**MSE—unchanged** up to a constant *[dead: 2nd-critic term]*	**Never called**	Welford-normalized transition distance, positive part, $0.01(1-p)$
MountainCar-v0	Symlog $\rightarrow$ EMA mean/var (α=0.005) $\rightarrow$ clip ±5	**Unchanged** *[dead: uncertainty-weighted min/max mixing]*	Adaptive expectile regression, weights $0.7/0.3 \rightarrow 0.5/0.5$	**Never called**	Online RND with random Fourier features, $0.2\,\text{relu}(z)(1-p)$
LunarLander-v3	Symmetric clip $[-20, 20]$	**Unchanged** *[dead: adaptive min/mean blend* and *the advertised adaptive-γ warm-up]*	Smooth-L1 (β=1)—a genuine live change from seed MSE	**Never called**	State-difference bonus (L2, cap 5.0), first 20% of training only, linearly decayed
Acrobot-v1	Clip ±10 + negligible decaying noise	**Unchanged** + live reward clamp ±1000 (inert for Acrobot) *[dead: optimistic min/max mixing]*	Asymmetric Huber, under/over weights $1.2/0.8$, $+10^{-5}Q^2$	**Never called**	Online RND (RFF) + NLMS predictor, EMA novelty, e^{-4p} decay

One consensus innovation survives across all three families, and it is the critic loss. Replacing MSE with a robust regression loss is reached independently under all three trainers: it is the SAC signature (16/17), it appears in 5/5 PPO champions (adaptive β in 3, median-derived adaptive δ in 1, fixed β=10 in 1) and in 3/4 DQN champions—**8 of the 9 PPO and DQN champions**, under two different trainers. A second, weaker consensus is symlog/clip-family observation encoding (SAC, 5/5 PPO, 4/4 DQN), but this slot's seed is a bare identity function, so nearly any change registers and the column is the least discriminating of the five. Beyond these two, the families are largely disjoint, and the disjointness is the informative part: each family's other innovations sit in the machinery specific to that family.

What this does and does not establish. The strong reading—that each family innovates in the slots characteristic of *that* family—holds cleanly for PPO and does not hold for DQN. But the DQN

shortfall is structural rather than empirical: the slots that would carry a characteristic DQN innovation are not exposed by the decomposition as implemented, and the value-target slot that *is* exposed degenerates to a fixed $\max_a Q'$ backup. The defensible claim is therefore that the decomposition supports genuine, family-appropriate innovation where the family has expressive room in the slots (SAC and PPO: five slots each) and degenerates where it does not (DQN: three live slots). We do *not* claim the three families have equal searchable surface area; DQN is materially narrower.

Caveats specific to the cross-algorithm champions. Three are worth stating plainly. **(1) Dead off-policy branches.** All four DQN champions carry unreachable twin-Q aggregation, GAE, entropy and second-critic terms, and three of the four rationale comments headline their Q-aggregation as the main change. **(2) LunarLander's advertised adaptive-γ warm-up is dead**, sitting inside the never-executed twin-Q branch; the live target uses plain γ. This is the most tempting mis-attribution in the set. **(3) One PPO champion carries an entropy term with a sign opposite to the trainer's.** Our trainer's own convention contributes $+\text{coef} \cdot \overline{\log \pi}$ to the loss, which maximises entropy. The gym-Hopper-v5 champion instead adds $-\text{coef} \cdot \overline{\log \pi}$, an entropy *penalty* rather than a bonus, and still beats its baseline. The gym-HalfCheetah-v5 and gym-Walker2d-v5 champions write the bonus through double negatives but are algebraically identical to the trainer's convention; we flag only the Hopper champion, in Table 15. Finally, these are the nine best-run champions only; we make no claim that these component choices recur across the other evolution seeds of the same task.