

# Repulsive Population Policy Gradients for Behaviorally Diverse Reinforcement Learning

Saeed Hedayatian<sup>1</sup> Harsh Shah<sup>1</sup> Celine Mneimneh<sup>2</sup> Stefanos Nikolaidis<sup>1</sup>

## Abstract

The goal of Quality-Diversity Reinforcement Learning (QD-RL) is to find a population of high-performing policies that solve a decision-making problem in diverse ways. Despite recent progress, QD-RL methods predominantly rely on evolutionary operators—including black-box optimizers, mutations, and crossovers—to facilitate exploration and induce diversity. Scaling these methods becomes increasingly costly as policies and behavior spaces grow larger. Building on the Soft QD view of quality-diversity optimization, we introduce **Repulsive Population Policy Optimization (RePPO)**, which adapts PPO to jointly optimize a compact population of diverse, high-quality policies using a population-dependent, dynamically shaped reward. On standard continuous-control tasks, RePPO achieves a strong quality-diversity tradeoff compared with state-of-the-art QD-RL methods. Ablations further characterize this tradeoff by examining how each component affects individual quality and collective diversity. To assess RePPO’s scalability, we extend our evaluation to learning from pixels and fine-tuning large language models. In pixel-based tasks, where large policies make most traditional QD-RL methods infeasible, RePPO maintains quality on par with standard PPO while achieving substantially higher diversity. Its successful application to language model fine-tuning further demonstrates, for the first time, that QD-RL methods can scale to this domain. Together, these results show that directly optimizing compact populations with RePPO provides a scalable approach to QD-RL across increasingly complex domains.

## 1. Introduction

Standard RL algorithms typically seek one policy that maximizes expected return, even though many tasks admit qualitatively different strategies. A humanoid can move forward by walking, hopping, or dragging itself (Batra et al., 2024); a manipulator can reach an object from different angles (Huber et al., 2023; 2025), and an autonomous vehicle can make progress while expressing different driving tendencies (Shiroshita et al., 2020; Zhang et al., 2022). Retaining these strategies allows us to compare how different behaviors achieve high reward and makes the available alternatives explicit. Quality-Diversity (QD) optimization formalizes this broader objective by seeking solutions that combine high task performance with diversity under a user-defined behavior descriptor (Pugh et al., 2016). The descriptor specifies which aspects of behavior should distinguish the solutions, making the desired diversity an explicit part of the learning problem.

The scalability of QD-RL depends on both how diverse policies are discovered and how the resulting collection is represented. The dominant approach, exemplified by MAP-Elites (Cully et al., 2015), partitions behavior space into cells and stores the highest-quality solution found in each cell. This makes behavioral coverage explicit, but ties the representation of diversity to a discretization: finer grids create more cells to fill, and the number of cells grows rapidly with descriptor dimension. These costs become substantial when each policy is a large neural network. Recent methods have improved search efficiency by incorporating deep RL updates (Pierrot et al., 2022; Faldor et al., 2023; Batra et al., 2024), yet diversity remains organized around an archive and often a separate evolutionary search procedure. We therefore ask whether quality and diversity can instead be optimized directly over a compact population using standard RL updates.

A continuous objective over a finite population offers a way to pursue this goal without organizing the search around archive cells. We build on Soft Quality-Diversity optimization (Hedayatian & Nikolaidis, 2026b), which measures the behavioral coverage induced by the population through smooth contributions from individual solutions. A high-quality solution contributes strongly around its descrip-

<sup>1</sup>Thomas Lord Department of Computer Science, University of Southern California, Los Angeles, USA <sup>2</sup>Independent Researcher. Correspondence to: Saeed Hedayatian <saeedhed@usc.edu>.

tor, while nearby solutions provide overlapping coverage. Soft QD captures this redundancy through a tractable lower bound that rewards individual quality and penalizes high-quality pairs that are behaviorally close. The population size can then be chosen directly, with diversity expressed through interactions among its members. Applying this objective to RL, however, requires optimizing both return and behavior without differentiating through the environment.

We show that standard policy gradients can optimize this population objective through a reward shaped by the current population. The key observation is that additive Markovian descriptors (Batra et al., 2024) express behavior as the expected cumulative value of per-step signals. A weighted combination of these signals can therefore serve as a reward for moving the behavior in a chosen direction. We establish that, at the current population, each policy’s gradient of the Soft QD lower bound is exactly a policy gradient under a shaped reward combining task performance with a population-derived behavior term. This connection yields **Repulsive Population Policy Optimization (RePPO)**, which adapts PPO (Schulman et al., 2017) to joint population training. RePPO uses this formulation as a starting point, adapting its task and diversity terms for stable PPO training while preserving a consistent task-improvement signal.

Our evaluation examines both the quality-diversity trade-off under a fixed interaction budget and the scalability of direct population training. Standard continuous-control tasks provide a comparison with established archive-based QD-RL methods. Atari games extend this evaluation to learning directly from pixels in a standard discrete-action domain. Lastly, zero-RL fine-tuning of LLMs on mathematical reasoning tasks further tests the approach in large language model reinforcement learning. Component ablations complement these experiments by examining how the algorithm’s design choices affect training and the balance between individual quality and collective diversity.

Our contributions are:

1. We derive a population-dependent shaped reward whose policy gradient matches each policy’s gradient of the Soft QD lower bound for additive Markovian behavior descriptors. This connects a continuous QD objective to standard policy optimization in RL.
2. We introduce RePPO, a practical extension of PPO that jointly trains a compact policy population without behavior-space discretization or evolutionary operators. We develop controls for stable population training and ablate their role in balancing quality and diversity.
3. We show competitive quality-diversity performance on continuous-control benchmarks, and demonstrate scalability to pixel-based Atari policies and RL fine-tuning of LLMs. The language model experiments

provide the first demonstration of QD-RL for large language model fine-tuning.

## 2. Related Work

### 2.1. Quality-Diversity objectives and search

QD research has developed both representations of behavioral coverage and search procedures for discovering high-quality solutions. MAP-Elites (Cully et al., 2015) represents coverage through a partition of behavior space and retains the highest-quality solution found in each region. This separates the task objective from the behavioral dimensions along which diversity is sought (Pugh et al., 2016). Alternative archive structures address limitations of regular grids; for example, centroidal Voronoi tessellations allow the number of archive regions to be chosen independently of descriptor dimension (Vassiliades et al., 2017; Mouret, 2023).

Adaptive search and gradient information have substantially improved the efficiency of this archive-based framework. CMA-ME combines covariance-adaptive search with archive improvement (Fontaine et al., 2020), while CMA-MAE introduces adaptive acceptance thresholds to improve exploration (Fontaine & Nikolaidis, 2023). Scalable variants reduce the cost of covariance adaptation for neural controllers (Tjanaka et al., 2023a). Differentiable QD uses gradients of both quality and behavior descriptors to guide search in the joint objective-behavior space (Fontaine & Nikolaidis, 2021). These advances show how the structure of a QD problem can inform its search operators.

Soft QD provides a continuous population objective that supports direct gradient-based optimization (Hedayatian & Nikolaidis, 2026b). Its SQUAD algorithm optimizes a tractable lower bound through quality-weighted interactions between population members. We inherit this objective and develop policy-gradient estimators and a practical RL training procedure.

### 2.2. Quality-Diversity in reinforcement learning

QD-RL has advanced through both scalable evolutionary search and the incorporation of deep RL updates. ME-ES scales archive-based search to neural policies using evolution strategies (Colas et al., 2020). PGA-ME combines genetic variation with off-policy actor gradients (Nilsson & Cully, 2021). DCG-ME extends this approach with a descriptor-conditioned critic and distills archived behaviors into a single descriptor-conditioned policy (Faldor et al., 2023). DCRL-ME further advances this line of methods by injecting specialized versions of the descriptor-conditioned policy into the offspring population (Faldor et al., 2025). QD-PG combines quality and diversity policy gradients to improve exploration and sample efficiency (Pierrot et al.,

2022). These methods demonstrate different ways to use experience from policy evaluation to guide the QD search.

On-policy optimization and parallel execution provide further routes to improving QD-RL. Work on approximating objective and descriptor gradients brought differentiable QD to RL settings (Tjanaka et al., 2022). PPGA uses PPO to estimate these gradients through Markovian measure proxies, and an evolutionary search over gradient coefficients guides archive improvement (Batra et al., 2024). RePPO builds on the use of RL updates for quality and behavior, with population interactions determined by a continuous QD objective.

QD-RL also includes approaches that represent diverse behaviors through a conditioned policy. QDAC, for instance, jointly learns high-performing skills using successor-feature critics under behavioral constraints (Grillotti et al., 2024). Its skill-conditioned actor provides a shared representation of many behaviors. RePPO instead treats a finite population of policies as the optimization variable with interactions derived from the Soft QD lower bound.

### 2.3. Diverse policies and behavior representations

Direct policy-population optimization has also been studied through diversity objectives outside the QD formulation. DOMiNO optimizes occupancy-based diversity while maintaining per-policy near-optimality constraints, and expresses diversity gradients through intrinsic rewards (Zahavy et al., 2022). DvD uses a determinant-based population diversity criterion (Parker-Holder et al., 2020), while DIPG encourages differences between policy-induced trajectory distributions using maximum mean discrepancy (Masood & Doshi-Velez, 2019). RePPO’s distinction lies in deriving quality-weighted interactions from a QD objective over given behavior descriptors.

Skill discovery and descriptor learning address complementary aspects of behavioral diversity. DIAYN learns distinguishable skills through an information-theoretic objective (Eysenbach et al., 2018), APS combines exploration with successor-feature learning (Liu & Abbeel, 2021), and METRA develops a metric-aware objective for scalable unsupervised skill discovery (Park et al., 2024). Moreover, connections between skill discovery and goal-conditioned RL can provide further approaches to learning behavioral alternatives (Choi et al., 2021). Lastly, unsupervised QD approaches address the representation of diversity by constructing behavior descriptors from sampled trajectories (Grillotti & Cully, 2022; Hedayatian & Nikolaidis, 2026a). Our focus is the population optimization problem given a descriptor set. The policy-gradient construction applies when that descriptor admits the additive representation specified below, independent of how they are obtained.

## 3. Background

### 3.1. The Soft QD population objective

We adopt the tractable population objective derived in Soft QD (Hedayatian & Nikolaidis, 2026b). Let  $\Theta$  be a solution space, with quality function  $f : \Theta \rightarrow \mathbb{R}^+$  and behavior descriptor  $\mathbf{b} : \Theta \rightarrow \mathbb{R}^d$ . For a population  $\theta = \{\theta_1, \dots, \theta_N\}$ , write  $f_n = f(\theta_n)$  and  $\mathbf{b}_n = \mathbf{b}(\theta_n)$ . In our setting, each  $\theta_n$  parameterizes a policy.

With a positive multiplicative constant omitted and the interaction scale denoted by  $\ell > 0$ , the Soft QD lower bound gives the following maximization objective

$$S(\theta) = \sum_{n=1}^N f_n - \sum_{1 \leq i < j \leq N} \sqrt{f_i f_j} \exp\left(-\frac{\|\mathbf{b}_i - \mathbf{b}_j\|_2^2}{\ell^2}\right). \quad (1)$$

The first term sums individual qualities while the second penalizes behavioral overlap between pairs, with the penalty increasing when both policies have high quality. The scale  $\ell$  determines how far these interactions extend in behavior space. Our goal is to optimize this inherited objective using RL updates.

### 3.2. RL with additive behavior descriptors

In RL, both quality and behavior can be represented through expected cumulative signals (Silver et al., 2021; Batra et al., 2024). Consider an MDP (Puterman, 2014)  $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \rho_0, \gamma)$  and a stochastic policy  $\pi_\theta(a | s)$ . A trajectory  $\tau = (s_0, a_0, s_1, a_1, \dots)$  is generated from  $\rho_0$ , the policy, and the transition kernel  $P$ . We assume that the following expectations are finite, with sums ending at termination in episodic tasks.

The policy’s quality is its expected discounted task return,

$$f(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[ \sum_t \gamma^t r(s_t, a_t) \right]. \quad (2)$$

We consider additive Markovian behavior descriptors of the form

$$\mathbf{b}(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[ \sum_t \gamma^t \mathbf{m}(s_t, a_t) \right], \quad (3)$$

where  $\mathbf{m} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}^d$  is a vector of per-step behavior signals (cumulants). Each coordinate describes an aspect of behavior through its expected cumulative value. The gradient computation presented next concerns such descriptors.

### 3.3. Policy gradients for quality and behavior

The same policy-gradient identity provides gradients for task return and for directional changes in behavior. For a

scalar cumulant  $c(s, a)$ , define

$$J_c(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[ \sum_t \gamma^t c(s_t, a_t) \right]. \quad (4)$$

Let  $Q_c^{\pi_\theta}(s, a)$  denote its expected future discounted sum after taking action  $a$  in state  $s$ . Under the usual regularity assumptions, the policy-gradient theorem (Williams, 1992) gives

$$\nabla_\theta J_c(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[ \sum_t \gamma^t \nabla_\theta \log \pi_\theta(a_t | s_t) Q_c^{\pi_\theta}(s_t, a_t) \right]. \quad (5)$$

Consequently, for any fixed vector of coefficients  $\mathbf{v} \in \mathbb{R}^d$ ,

$$(\nabla_\theta \mathbf{b}(\theta))^\top \mathbf{v} = \nabla_\theta J_{\mathbf{v}^\top \mathbf{m}}(\theta). \quad (6)$$

A directional descriptor gradient is thus an ordinary policy gradient under the scalar reward  $\mathbf{v}^\top \mathbf{m}(s, a)$ . This identity estimates behavioral changes from sampled experience without differentiating through the environment. Section 4.1 uses it to determine reward coefficients from the current population.

## 4. Repulsive Population Policy Optimization

RePPO jointly trains a population by using the policies' relative qualities and behaviors to shape their PPO updates. Each iteration collects experience, summarizes the population, and determines how each policy should change its behavior to complement the others. These directions become weights on behavior advantages, which are combined with the task advantage in the actor update. We first derive this connection from the Soft QD objective, then explain how RePPO constructs the population interaction and incorporates it into PPO.

### 4.1. Population gradients as shaped rewards

The Soft QD gradient balances individual task improvement against behavioral overlap with other high-quality policies. For policy  $k$ , let  $W_k$  denote the objective's marginal value of increasing its quality and let  $\mathbf{c}_k$  denote its direction of improvement in behavior space. With independently parameterized policies, differentiating Eq. (1) and applying the chain rule separates these contributions:

$$\nabla_{\theta_k} S(\theta) = W_k \nabla_{\theta_k} f_k + (\nabla_{\theta_k} \mathbf{b}_k)^\top \mathbf{c}_k, \quad (7)$$

with the coefficients defined as

$$W_k = 1 - \frac{1}{2} \sum_{i \neq k} \sqrt{\frac{f_i}{f_k}} \exp\left(-\frac{\|\mathbf{b}_k - \mathbf{b}_i\|_2^2}{\ell^2}\right), \quad (8)$$

$$\mathbf{c}_k = \sum_{i \neq k} \frac{2\sqrt{f_k f_i}}{\ell^2} \exp\left(-\frac{\|\mathbf{b}_k - \mathbf{b}_i\|_2^2}{\ell^2}\right) (\mathbf{b}_k - \mathbf{b}_i). \quad (9)$$

The vector  $\mathbf{c}_k$  combines directions away from other policies, weighted by both policies' qualities and their behavioral overlap. It therefore encourages behavioral separation where high-quality policies overlap. Each pairwise contribution vanishes at exact coincidence and decays at large separation.

Additive behavior descriptors turn this population interaction into an ordinary RL reward.

**Proposition 4.1.** *Assume the additive representation of Section 3.2, independently parameterized policies with strictly positive qualities, and the usual policy-gradient regularity conditions. At the current population, the gradient of  $S$  (Equation 1) with respect to  $\theta_k$  is the policy gradient under*

$$\tilde{r}_k(s, a) = W_k r(s, a) + \mathbf{c}_k^\top \mathbf{m}(s, a), \quad (10)$$

with  $W_k$  and  $\mathbf{c}_k$  held fixed when differentiating the return under this reward.

The result follows by applying the policy-gradient theorem to both terms in (7). Appendix A provides the proof. Each policy thus receives its own scalar reward, whose behavior component depends on the current population. The resulting update can be estimated from sampled experience without differentiating through the environment. As the population evolves, recomputing the coefficients changes how quality and diversity enter each policy's update.

The exact quality coefficient can nevertheless conflict with task learning. When  $W_k < 0$ , the objective encourages a policy to become *less efficient* in order to change its behavior and occupy a different behavioral niche. In RL, we found such reversal of the task-learning signal to cause extreme instabilities that leads policies towards degenerate behaviors from which they can't easily recover. This phenomenon is more thoroughly studied in Appendix D.1. As such, RePPO fixes the task coefficient to one and retains the derived repulsion structure, with practical controls described next.

### 4.2. Constructing diversity weights from the population

RePPO converts population repulsion into a diversity-weight vector  $\mathbf{p}_k$  for each policy. Its components will weight the advantages of the original behavior signals: positive and negative weights add pressure to increase and decrease the corresponding features. Constructing this vector requires estimating the population's qualities and behaviors, computing repulsion in a chosen behavior coordinates, and expressing repulsion in terms of the signals used for learning.

Episode-level summaries provide a smoothed description of the population. For each policy, we maintain exponential moving averages  $\hat{F}_k$  of completed-episode returns  $\sum_t r_t$  and  $\hat{\mathbf{d}}_k$  of episode-average descriptors  $T_\tau^{-1} \sum_t \mathbf{m}_t$ , where  $T_\tau$  is the episode length. The returns measure individual quality, while the descriptors locate policies in behavior

space. Smoothing reduces fluctuations in both the interaction strengths and directions as policies learn. We obtain positive quality weights using  $f_k^+ = \hat{F}_k - \min_j \hat{F}_j + \epsilon_f$ , with  $\epsilon_f > 0$ .

Behavioral distances are measured in transformed coordinates. Following SQUAD (Hedayatian & Nikolaidis, 2026b), we map descriptors normalized to  $[-1, 1]^d$  into unbounded coordinates using the componentwise logit  $g(d) = \log((1+d)/(1-d))$ . This defines the interaction geometry and stretches differences near the descriptor bounds. We compute local repulsion  $\mathbf{c}_k^{\text{local}}$  using (9), replacing  $(f_i, \mathbf{b}_i)$  with  $(f_i^+, g(\hat{\mathbf{d}}_i))$  and restricting the sum to the  $K$  nearest other policies in transformed space. In the experiments, we use  $K=8$  unless otherwise stated. The neighborhood selects which policies interact, while the bandwidth  $\ell$  determines how their influence decays with distance. This gives us a direction in the (transformed) behavior space that the policy should follow to maximize behavioral coverage (i.e., diversity).

The resulting direction must be converted into weights on the original behavior signals before it can enter PPO. Repulsion is computed in logit coordinates, but the behavior advantages describe changes in the original features  $\mathbf{m}_t$ . The logit Jacobian  $J_g$  connects these coordinates through the chain rule. After this conversion, we cap the vector’s norm to control the overall diversity pressure:

$$\mathbf{p}_k = \text{cap}_{C_{\max}} \left( J_g(\hat{\mathbf{d}}_k)^\top \mathbf{c}_k^{\text{local}} \right), \quad (11)$$

where the cap preserves direction, rescales vectors exceeding norm  $C_{\max}$  to that norm, and leaves all others unchanged.

### 4.3. PPO updates with population-shaped advantages

Separate task and behavior critics allow the population interaction to change while keeping each critic tied to a fixed signal. For each policy, a task critic and a behavior critic with  $d$  outputs estimate discounted generalized advantages (Schulman et al., 2016) for  $r_t$  and each  $m_{t,j}$ . The population affects how the channels are combined, without entering the critics’ prediction targets.

RePPO combines these learning signals after putting their advantage channels on comparable scales. Within each policy’s PPO minibatch, we standardize each channel separately, denoting the resulting advantages by  $\bar{A}$ . This prevents differences in the numerical scales of the channels from determining their relative influence. The actor then uses

$$\tilde{A}_{k,t} = \bar{A}_{k,t}^T + \sum_{j=1}^d p_{k,j} \bar{A}_{k,t}^{m_j}. \quad (12)$$

The task channel retains unit weight, while  $\mathbf{p}_k$  determines the contribution of each behavior channel, collectively

capped by  $C_{\max}$ . The combined advantage is then used with the clipped PPO actor update (Schulman et al., 2017). The critics are fitted to return targets for their respective signals. Note that setting  $\mathbf{p}_k = \mathbf{0}$  recovers the task-only PPO actor update.

Updating population interaction coefficients and policy learning alternate throughout training. After collecting rollouts from all policies, RePPO updates the population summaries and computes the diversity weights. These weights remain fixed throughout the PPO epochs on that rollout data, so each policy learns consistently during its update. The updated policies then generate new experience, refreshing the summaries and the interaction for the next iteration. Algorithm 1 summarizes this loop. The episode statistics, unit task coefficient, local and capped interactions, and channel normalization modify the exact gradient in Proposition 4.1 to enable stable and efficient optimization. We examine their effects, together with population size and other hyperparameters, in Appendix D.

## 5. Experiments

Our experiments test whether directly optimizing a compact population yields a favorable balance of quality and diversity, and whether this approach scales to domains where archive-based QD-RL becomes impractical. We compare RePPO with established QD-RL methods on continuous-control benchmarks, then turn to two settings where archives of policies become costly or impractical: pixel-based Atari policies and LLM fine-tuning for mathematical reasoning. Appendices G and H add results on pixel-based locomotion and on LunarLander.

**Benchmarks.** The continuous-control benchmark consists of four Brax locomotion tasks (Freeman et al., 2021) commonly used in QD-RL: Ant, HalfCheetah, Humanoid, and Walker2D (Batra et al., 2024). Fitness, the quality  $f$  of Section 3.2, is the episode return, and the descriptor records the fraction of time each foot touches the ground, giving four dimensions for Ant and two for the other tasks. For Atari, we use four games from the Arcade Learning Environment (Bellemare et al., 2015; Machado et al., 2018): Boxing, Asterix, Seaquest, and Ms. Pac-Man. For each task, fitness is the game score, and two-dimensional descriptors record where the agent plays, its distant to opponent (in Boxing), and the order in which it clears the maze (in Ms. Pac-Man). For LLMs, we fine-tune two base models, Qwen3-1.7B (Team, 2025) and Granite-4.1-3B (IBM Granite Team, 2026), on MATH problems (Hendrycks et al., 2021) with a binary correctness reward, and evaluate them on MATH-500 (Lightman et al., 2024), Minerva (Lewkowycz et al., 2022), and OlympiadBench (He et al., 2024). Four descriptors characterize how a correct solution is written: the share

**Algorithm 1** Repulsive Population Policy Optimization (RePPO)

---

**Require:** Population size  $N$ ; interaction parameters  $K, \ell^2, C_{\max}, \epsilon_f$ ; EMA rates; PPO settings

- 1: Initialize policies, task critics, and behavior critics
- 2: Initialize  $\hat{F}_k \leftarrow \epsilon_f$  and  $\hat{\mathbf{d}}_k \leftarrow \mathbf{0}$  for all  $k$
- 3: **for** each training iteration **do**
- 4:   **for** each policy  $k$  **do**
- 5:     Collect on-policy data  $\mathcal{D}_k$ , including task and behavior signals
- 6:     Update  $\hat{F}_k, \hat{\mathbf{d}}_k$  from completed episodes
- 7:   **end for**
- 8:    $f_k^+ \leftarrow \hat{F}_k - \min_j \hat{F}_j + \epsilon_f$  for all  $k$
- 9:   Transform smoothed descriptors using the logit and boundary clipping
- 10:   **for** each policy  $k$  **do**
- 11:     Compute local repulsion using shifted qualities and nearest neighbors in transformed space
- 12:     Convert and cap this repulsion to obtain  $\mathbf{p}_k$  using (11)
- 13:     Compute task and behavior GAE channels on  $\mathcal{D}_k$
- 14:   **end for**
- 15:   **for** each policy  $k$  and each PPO epoch **do**
- 16:     **for** each minibatch from  $\mathcal{D}_k$  **do**
- 17:       Standardize each advantage channel within the minibatch
- 18:       Combine channels using  $\mathbf{p}_k$  as in (12)
- 19:       Apply the PPO actor update and fit the critics to their return targets
- 20:     **end for**
- 21:   **end for**
- 22: **end for**
- 23: **return**  $\{\pi_{\theta_k}\}_{k=1}^N$

---

of Python *code*, the rate of explicit self-*verification*, the share of  $\text{\LaTeX}$  *symbols*, and the use of *markdown* structure. In this domain, all descriptors and the reward are sparse as only the last token may get non-zero values. Appendices C, E, and F define them and list all hyperparameters.

**Baselines.** In every domain, we compare against Independent PPO (Ind-PPO), which uses RePPO’s implementation, population size, and hyperparameters with the diversity weights  $\mathbf{p}_k$  set to zero, so that the comparison isolates the effect of repulsion. It is equivalent to training  $N$  PPO policies each with a different seed. In continuous control, we additionally compare against archive-based methods representing the main families of QD-RL: PPGA (Batra et al., 2024), which uses PPO for differentiable QD; PGA-ME (Nilsson & Cully, 2021) and DCRL-ME (Faldor et al., 2025), which add TD3-based variation (Fujimoto et al., 2018) to MAP-Elites; and the black-box Sep-CMA-MAE (Tjanaka et al.,

2023a). In Atari, since the aforementioned methods become prohibitively inefficient and costly, we compare against ReQD (Wang et al., 2024), which shares one representation network across its archive. For LLMs, we compare against the base model and two prompting baselines in which each one of the members is the base model with a different style instruction, either combining extreme values of several descriptors or emphasizing a single one.

**Evaluation.** After training, we evaluate every policy anew to estimate its fitness and mean descriptor, using 256 episodes in continuous control, 32 full games in Atari, and 8 samples per test problem for LLMs. Archive-based methods return thousands of policies, many of which fail to solve the task. Because of this, we discard policies whose fitness is below 10% of the best policy on that task and evaluate the remaining policies. On Ant, we use a 25% threshold, since stationary policies with varied contact patterns collect enough survival bonus to clear the lower bar. We measure quality by the mean fitness of the retained policies and diversity by the Vendi Score of their descriptors (Friedman & Dieng, 2023), which can be read as the effective number of distinct behaviors in the population. Their product, the quality-weighted Vendi Score (qVS) (Nguyen & Dieng, 2024), serves as a combined measure of quality and diversity. We also report each population’s best fitness before filtering, and vary the threshold in Appendix C.4.

### 5.1. Continuous control

RePPO trains  $N = 32$  policies for 2.05 B environment steps in total, the same budget as Ind-PPO, PGA-ME, DCRL-ME, and Sep-CMA-MAE, while PPGA uses 2.74 B steps. PPGA’s budget can only be controlled indirectly through its number of iterations which is why it has a higher budget.

RePPO attains the highest qVS on all four tasks by combining the quality of PPO training with diversity that rivals archive-based search (Figure 1, top). The baselines fall on either side of this balance. Ind-PPO trains strong policies, but without repulsion they converge to the same behavior. Archive-based methods cover behavior space broadly, but with their budget spread over thousands of cells, most of their policies remain weaker. RePPO’s population balances the two. On Humanoid, it is about twice as diverse as any archive at a mean fitness close to that of the fittest one. Ant proves to be the hardest task for QD baselines, with many of them struggling to find high-performing policies and instead diversifying low-quality behaviors. This is particularly evident when we look at the highest fitness among the populations that each method finds, where these methods trail RePPO and Ind-PPO substantially (Table 4). Still, RePPO is able to find high-quality policies while diversifying more than Ind-PPO. Overall, DCRL-ME comes closest and even matches RePPO on HalfCheetah.

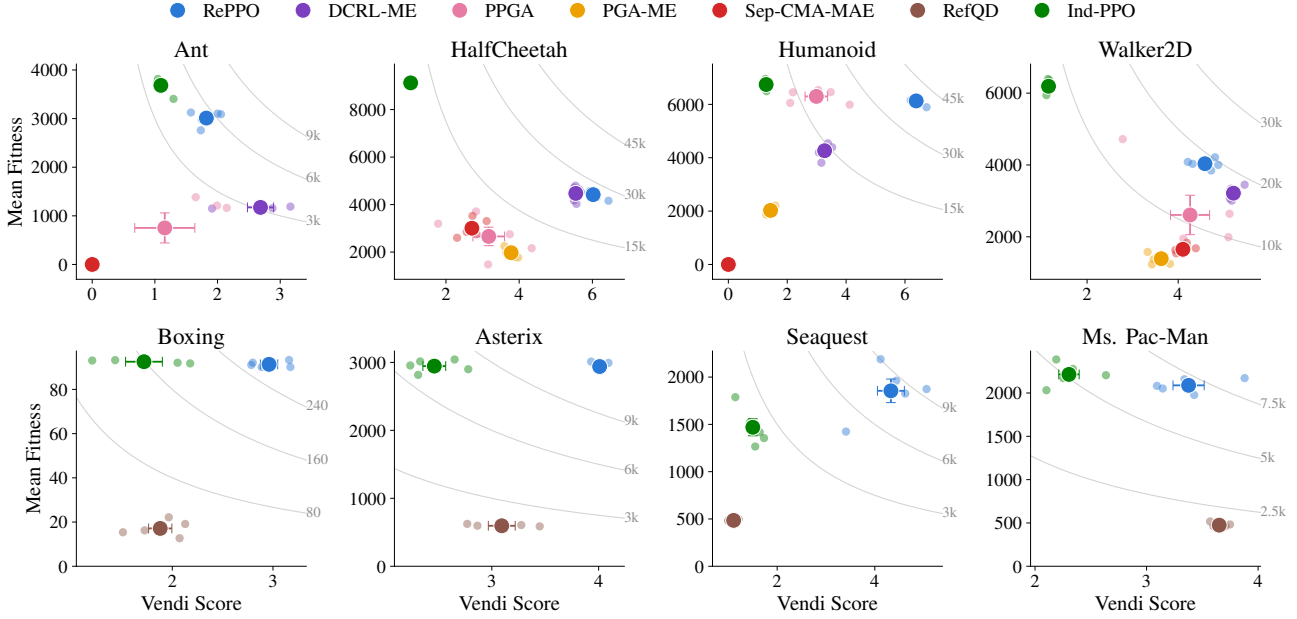


Figure 1. Quality and diversity of the final populations in continuous control (top) and Atari (bottom). Large markers show the mean over five seeds with one standard error, and faint dots show individual runs. Gray curves are levels of constant qVS, signifying the combined quality and diversity of a population. Up and to the right is better.

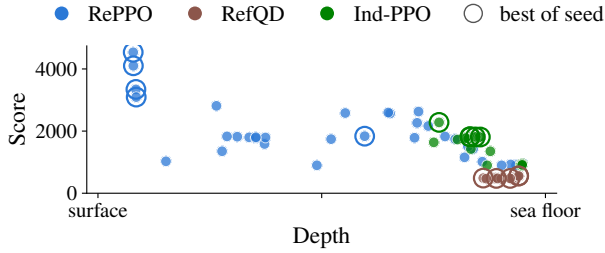


Figure 2. Score and mean depth of every Seaquest policy over five seeds that passes the 10% quality filter, with the best policy of each seed circled. The policies of Ind-PPO and RefQD stay near the sea floor, whereas each RePPO population spreads across depths, and its best policies operate just below the surface.

The results show that repulsion costs part of the population’s mean fitness but leaves its best policies largely intact. Policies pushed toward less common contact patterns adopt gaits that are often slower, which lowers the average, most visibly on HalfCheetah. The best policy of each population, however, stays close to the best one trained by Ind-PPO and outperforms those of nearly all archive-based baselines (Table 4). Appendices C.4 and D visualize the resulting populations and examine how each component of RePPO shapes this tradeoff.

## 5.2. Pixel-based policies in Atari

RePPO and Ind-PPO train  $N = 8$  policies for 10 M steps each, following the standard PPO configuration for

Atari (Huang et al., 2022). RefQD uses a comparable budget of 86.4 M steps and maintains an archive of 1000 cells. Its training configuration and hyperparameters largely follow the published code.

RePPO retains the quality of independent PPO training while making its populations substantially more diverse (Figure 1, bottom). Its mean scores are close to or above Ind-PPO’s in every game and its diversity is higher than both baselines, culminating in substantial gains in qVS over both baselines. RefQD’s archives cover behavior space broadly, but their policies score far below those trained with PPO.

Diversity can also uncover strategies that independent training misses. In Seaquest, every policy trained by Ind-PPO, and every competent policy in RefQD’s archives, patrols near the sea floor (Figure 2). Repulsion pushes some of RePPO’s policies toward shallower water, and in four of the five seeds, the best policy of the population operates just below the surface, where it outscores every Ind-PPO policy. Here, diversity improves not only the population as a whole but also its best member. Appendix E.4 visualizes the behaviors in all four games.

## 5.3. Fine-tuning large language models

In fine-tuning large language models, each population consists of  $N = 5$  members, each a low-rank adapter (Hu et al., 2022) on a shared frozen backbone with its own adapted critic. Members are trained with PPO in the zero-RL set-

Table 1. LLM populations on the test problems. Vendi is the Vendi Score of the members’ mean descriptors, and ID is the accuracy of a classifier that identifies a solution’s author (chance is 20%).

| Model   | Method            | pass@1       | Vendi       | ID         |
|---------|-------------------|--------------|-------------|------------|
| Qwen3   | Base model        | 0.320        | —           | —          |
|         | Prompt (corners)  | 0.274        | 1.11        | 51%        |
|         | Prompt (emphasis) | 0.275        | 1.08        | 44%        |
|         | Ind-PPO           | <b>0.423</b> | 1.04        | 41%        |
|         | RePPO             | 0.413        | <b>1.85</b> | <b>82%</b> |
| Granite | Base model        | 0.233        | —           | —          |
|         | Prompt (corners)  | 0.163        | 1.01        | 31%        |
|         | Prompt (emphasis) | 0.194        | 1.03        | 37%        |
|         | Ind-PPO           | <b>0.560</b> | 1.03        | 53%        |
|         | RePPO             | 0.550        | <b>2.57</b> | <b>81%</b> |

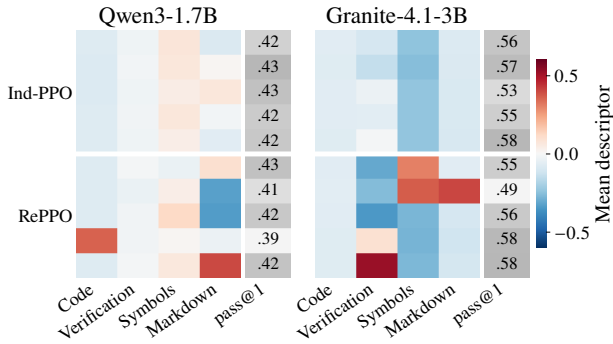


Figure 3. Mean descriptor of each member over its test solutions, alongside its pass@1. Descriptors are centered on the base model’s style, and incorrect solutions count as zero. RePPO’s members adopt distinct styles while matching the accuracy of Ind-PPO’s near-identical members.

ting (Hu et al., 2025) for 100 iterations of 1024 solutions each. We select checkpoints by validation accuracy, and report behavior and performance over test problems (Appendix F).

RePPO’s members acquire distinct and consistent writing styles while nearly matching the accuracy of independent training (Table 1). In both models, repulsion makes the members’ descriptors far more diverse than those of Ind-PPO or of either prompting baseline. The members are also distinguishable from their text alone: a simple classifier identifies which RePPO member wrote a solution much more reliably than for any other population. The price is a small but consistent drop of about one point in per-member accuracy, while the population as a whole solves as many problems as Ind-PPO’s (Appendix F.4). Prompting, in contrast, changes the members’ style only slightly and at a much higher cost of accuracy.

The styles differ across members and base models (Figure 3). On Qwen3, one RePPO member appends Python code to its solutions, another writes in itemized markdown, and two write in plain prose. On Granite, the members separate

along verification and notation: some re-check their intermediate results far more often than others, and some typeset their reasoning in  $\LaTeX$  instead of plain text. Appendix F.5 provides examples of the different models in RePPO’s population generating stylistically diverse responses to the same problems.

## 6. Conclusion and Outlook

We introduced RePPO, a Quality-Diversity algorithm that directly trains a compact population of reinforcement learning policies. Starting from the Soft QD objective, we showed that additive Markovian behavior descriptors turn each policy’s population gradient into an ordinary policy gradient under a shaped reward, which RePPO optimizes with PPO. On continuous-control benchmarks, RePPO attains a better balance of quality and diversity than established QD-RL methods. Because its cost scales with the population size rather than with the resolution of an archive, it extends to pixel-based Atari policies and to LLM fine-tuning, where it makes populations substantially more diverse at nearly the quality of independent training.

RePPO is designed for optimizing compact populations of useful policies rather than exhaustive coverage of behavior space, for which archive-based methods remain better suited. It also requires descriptors that admit additive forms, and it diversifies whatever these descriptors measure: in our LLM experiments, hand-designed descriptors produced differences mostly in presentation rather than in strategy. Addressing these limitations calls for learned or preference-derived descriptors, adaptive bandwidths and population sizes, and hybrids with external archives.

More broadly, shaping each policy’s reward by the population is useful whenever one optimum is not enough, from adaptation to changing conditions to generative models whose adapters express different preferences or styles. In all these settings, diversity can be encoded as a reward induced by the geometry of a population.

## References

- Batra, S., Tjanaka, B., Fontaine, M., Petrenko, A., Nikolaidis, S., and Sukhatme, G. Proximal policy gradient arborescence for quality diversity reinforcement learning. In *International Conference on Learning Representations*, volume 2024, pp. 35884–35904, 2024.
- Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents (extended abstract). In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pp. 4148–4152. AAAI Press, 2015.

- Chalumeau, F., Lim, B., Boige, R., Allard, M., Grillotti, L., Flageat, M., Macé, V., Richard, G., Flajolet, A., Pierrot, T., et al. Qdax: A library for quality-diversity and population-based algorithms with hardware acceleration. *Journal of Machine Learning Research*, 25(108):1–16, 2024.
- Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H. P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating large language models trained on code. *CoRR*, abs/2107.03374, 2021. URL <https://doi.org/10.48550/arXiv.2107.03374>.
- Choi, J., Sharma, A., Lee, H., Levine, S., and Gu, S. S. Variational empowerment as representation learning for goal-conditioned reinforcement learning. In *International conference on machine learning*, pp. 1953–1963. PMLR, 2021.
- Colas, C., Madhavan, V., Huizinga, J., and Clune, J. Scaling map-elites to deep neuroevolution. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pp. 67–75, 2020.
- Cully, A., Clune, J., Tarapore, D., and Mouret, J. Robots that can adapt like animals. *Nat.*, 521(7553):503–507, 2015. doi: 10.1038/NATURE14422. URL <https://doi.org/10.1038/nature14422>.
- Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.
- Faldor, M., Chalumeau, F., Flageat, M., and Cully, A. Map-elites with descriptor-conditioned gradients and archive distillation into a single policy. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 138–146, 2023.
- Faldor, M., Chalumeau, F., Flageat, M., and Cully, A. Synergizing quality-diversity with descriptor-conditioned reinforcement learning. *ACM Transactions on Evolutionary Learning*, 5(1):1–35, 2025.
- Fontaine, M. and Nikolaidis, S. Differentiable quality diversity. *Advances in Neural Information Processing Systems*, 34:10040–10052, 2021.
- Fontaine, M. and Nikolaidis, S. Covariance matrix adaptation map-annealing. In *Proceedings of the genetic and evolutionary computation conference*, pp. 456–465, 2023.
- Fontaine, M. C., Togelius, J., Nikolaidis, S., and Hoover, A. K. Covariance matrix adaptation for the rapid illumination of behavior space. In *Proceedings of the 2020 genetic and evolutionary computation conference*, pp. 94–102, 2020.
- Freeman, C. D., Frey, E., Raichuk, A., Girgin, S., Mor-datch, I., and Bachem, O. Brax - A differentiable physics engine for large scale rigid body simulation. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021. URL <https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/d1f491a404d6854880943e5c3cd9ca25-Abstract-round1.html>.
- Friedman, D. and Dieng, A. B. The vendi score: A diversity evaluation metric for machine learning. *Transactions on Machine Learning Research*, 2023.
- Fujimoto, S., Hoof, H., and Meger, D. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pp. 1587–1596. PMLR, 2018.
- Grillotti, L. and Cully, A. Unsupervised behavior discovery with quality-diversity optimization. *IEEE Trans. Evol. Comput.*, 26(6):1539–1552, 2022. URL <https://doi.org/10.1109/TEVC.2022.3159855>.
- Grillotti, L., Faldor, M., León, B. G., and Cully, A. Quality-diversity actor-critic: Learning high-performing and diverse behaviors via value and successor features critics. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, pp. 16416–16459. PMLR / OpenReview.net, 2024. URL <https://proceedings.mlr.press/v235/grillotti24a.html>.
- He, C., Luo, R., Bai, Y., Hu, S., Thai, Z. L., Shen, J., Hu, J., Han, X., Huang, Y., Zhang, Y., Liu, J., Qi, L., Liu, Z., and Sun, M. Olympiadbench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 3828–3850. Association for Computational Linguistics, 2024. URL <https://doi.org/10.18653/v1/2024.acl-long.211>.

- Hedayatian, S. and Nikolaidis, S. AutoQD: Automatic discovery of diverse behaviors with quality-diversity optimization. In *The Fourteenth International Conference on Learning Representations*, 2026a. URL <https://openreview.net/forum?id=FNnJIf4ymV>.
- Hedayatian, S. and Nikolaidis, S. Soft quality-diversity optimization. In *The Fourteenth International Conference on Learning Representations*, 2026b. URL <https://openreview.net/forum?id=LCKixyFShR>.
- Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual, 2021*. URL <https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/be83ab3ecd0db773eb2dc1b0a17836a1-Abstract-round2.html>.
- Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL <https://openreview.net/forum?id=nZeVKeeFYf9>.
- Hu, J., Zhang, Y., Han, Q., Jiang, D., Zhang, X., and Shum, H. Open-reasoner-zero: An open source approach to scaling up reinforcement learning on the base model. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL [http://papers.nips.cc/paper\\_files/paper/2025/hash/ed873d79e7c268c020c4b4db13a2812a-Abstract-Conference.html](http://papers.nips.cc/paper_files/paper/2025/hash/ed873d79e7c268c020c4b4db13a2812a-Abstract-Conference.html).
- Huang, S., Dossa, R. F. J., Ye, C., Braga, J., Chakraborty, D., Mehta, K., and Araújo, J. G. M. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *J. Mach. Learn. Res.*, 23:274:1–274:18, 2022. URL <https://jmlr.org/papers/v23/21-1342.html>.
- Huber, J., H  l  non, F., Coninx, M., Amar, F. B., and Doncieux, S. Quality diversity under sparse reward and sparse interaction: Application to grasping in robotics. *arXiv preprint arXiv:2308.05483*, 2023.
- Huber, J., H  l  non, F., Kappel, M., de Loyola P  ez-Ubieta, I., Puente, S. T., Gil, P., Amar, F. B., and Doncieux, S. Qdgsset: A large scale grasping dataset generated with quality-diversity. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 01–08. IEEE, 2025.
- IBM Granite Team. Granite 4.1 LLMs: How they’re built. <https://huggingface.co/blog/ibm-granite/granite-4-1>, 2026.
- Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, pp. 611–626. ACM, 2023. URL <https://doi.org/10.1145/3600006.3613165>.
- Lewkowycz, A., Andreassen, A., Dohan, D., Dyer, E., Michalewski, H., Ramasesh, V. V., Slone, A., Anil, C., Schlag, I., Gutman-Solo, T., Wu, Y., Neyshabur, B., Gur-Ari, G., and Misra, V. Solving quantitative reasoning problems with language models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL [http://papers.nips.cc/paper\\_files/paper/2022/hash/18abbeef8cfe9203fdf9053c9c4fe191-Abstract-Conference.html](http://papers.nips.cc/paper_files/paper/2022/hash/18abbeef8cfe9203fdf9053c9c4fe191-Abstract-Conference.html).
- Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let’s verify step by step. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL <https://openreview.net/forum?id=v8L0pN6EOi>.
- Liu, H. and Abbeel, P. Aps: Active pretraining with successor features. In *International Conference on Machine Learning*, pp. 6736–6747. PMLR, 2021.
- Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M. J., and Bowling, M. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents (extended abstract). In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pp. 5573–5577. ijcai.org, 2018.
- Masood, M. A. and Doshi-Velez, F. Diversity-inducing policy gradient: Using maximum mean discrepancy to find a set of diverse policies. *arXiv preprint arXiv:1906.00088*, 2019.

- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533, 2015.
- Mouret, J.-B. Fast generation of centroids for map-elites. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, pp. 155–158, 2023.
- Nguyen, Q. and Dieng, A. B. Quality-weighted vendi scores and their application to diverse experimental design. In *Forty-first International Conference on Machine Learning*, 2024.
- Nilsson, O. and Cully, A. Policy gradient assisted map-elites. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 866–875, 2021.
- Park, S., Rybkin, O., and Levine, S. Metra: Scalable unsupervised rl with metric-aware abstraction. In *International Conference on Learning Representations*, volume 2024, pp. 18579–18603, 2024.
- Parker-Holder, J., Pacchiano, A., Choromanski, K. M., and Roberts, S. J. Effective diversity in population based reinforcement learning. *Advances in Neural Information Processing Systems*, 33:18050–18062, 2020.
- Pierrot, T., Macé, V., Chalumeau, F., Flajolet, A., Cideron, G., Beguir, K., Cully, A., Sigaud, O., and Perrin-Gilbert, N. Diversity policy gradient for sample efficient quality-diversity optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1075–1083, 2022.
- Pugh, J. K., Soros, L. B., and Stanley, K. O. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3:40, 2016.
- Puterman, M. L. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.
- Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL <http://jmlr.org/papers/v22/20-1364.html>.
- Schulman, J., Moritz, P., Levine, S., Jordan, M. I., and Abbeel, P. High-dimensional continuous control using generalized advantage estimation. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL <http://arxiv.org/abs/1506.02438>.
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- Shiroshita, S., Maruyama, S., Nishiyama, D., Castro, M. Y., Hamzaoui, K., Rosman, G., DeCastro, J., Lee, K.-H., and Gaidon, A. Behaviorally diverse traffic simulation via reinforcement learning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2103–2110. IEEE, 2020.
- Silver, D., Singh, S., Precup, D., and Sutton, R. S. Reward is enough. *Artificial intelligence*, 299:103535, 2021.
- Team, Q. Qwen3 technical report. *CoRR*, abs/2505.09388, 2025. URL <https://doi.org/10.48550/arXiv.2505.09388>.
- Tjanaka, B., Fontaine, M. C., Togelius, J., and Nikolaidis, S. Approximating gradients for differentiable quality diversity in reinforcement learning. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1102–1111, 2022.
- Tjanaka, B., Fontaine, M. C., Lee, D. H., Kalkar, A., and Nikolaidis, S. Training diverse high-dimensional controllers by scaling covariance matrix adaptation map-annealing. *IEEE Robotics and Automation Letters*, 8 (10):6771–6778, 2023a.
- Tjanaka, B., Fontaine, M. C., Lee, D. H., Zhang, Y., Balam, N. R., Dennler, N., Garlanka, S. S., Klapsis, N. D., and Nikolaidis, S. pyribs: A bare-bones python library for quality diversity optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2023, Lisbon, Portugal, July 15-19, 2023*, pp. 220–229. ACM, 2023b. URL <https://doi.org/10.1145/3583131.3590374>.
- Towers, M., Kwiatkowski, A., Balis, J. U., Cola, G. D., Deleu, T., Goulão, M., Andreas, K., Krimmel, M., KG, A., Perez-Vicente, R. D. L., Terry, J. K., Pierré, A., Schulhoff, S. V., Tai, J. J., Tan, H., and Younis, O. G. Gymnasium: A standard interface for reinforcement learning environments. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2026. URL <https://openreview.net/forum?id=qPMLvJxtPK>.
- van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pp. 2094–2100. AAAI Press, 2016. URL <https://doi.org/10.1609/aaai.v30i1.10295>.

- Vassiliades, V., Chatzilygeroudis, K., and Mouret, J.-B. Using centroidal voronoi tessellations to scale up the multi-dimensional archive of phenotypic elites algorithm. *IEEE Transactions on Evolutionary Computation*, 22(4):623–630, 2017.
- Wang, R.-J., Xue, K., Guan, C., and Qian, C. Quality-diversity with limited resources. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 51984–52001, 2024.
- Weng, J., Lin, M., Huang, S., Liu, B., Makoviichuk, D., Makoviychuk, V., Liu, Z., Song, Y., Luo, T., Jiang, Y., Xu, Z., and Yan, S. Envpool: A highly parallel reinforcement learning environment execution engine. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL [http://papers.nips.cc/paper\\_files/paper/2022/hash/8caaf08e49ddbada6694fae067442ee21-Abs tract-Datasets\\_and\\_Benchmarks.html](http://papers.nips.cc/paper_files/paper/2022/hash/8caaf08e49ddbada6694fae067442ee21-Abs tract-Datasets_and_Benchmarks.html).
- Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.
- Zahavy, T., Schroecker, Y., Behbahani, F., Baumli, K., Flennerhag, S., Hou, S., and Singh, S. Discovering policies with domino: Diversity optimization maintaining near optimality. *arXiv preprint arXiv:2205.13521*, 2022.
- Zeng, W., Huang, Y., Liu, Q., Liu, W., He, K., Ma, Z., and He, J. Simplerl-zoo: Investigating and taming zero reinforcement learning for open base models in the wild. *CoRR*, abs/2503.18892, 2025. URL <https://doi.org/10.48550/arXiv.2503.18892>.
- Zhang, Q., Gao, Y., Zhang, Y., Guo, Y., Ding, D., Wang, Y., Sun, P., and Zhao, D. Trajgen: Generating realistic and diverse trajectories with reactive and feasible agent behaviors for autonomous driving. *IEEE Transactions on Intelligent Transportation Systems*, 23(12):24474–24487, 2022.

## A. Derivations

This section proves Proposition 4.1 and then traces how RePPO’s update follows from it. We use the notation of Sections 3.2 and 4:  $f_k$  and  $\mathbf{b}_k$  are the quality and descriptor of policy  $k$ ,  $\mathbf{m}$  is the vector of per-step behavior signals, and  $\ell$  is the interaction scale.

### A.1. Proof of Proposition 4.1

We restate the policy-gradient identity of Section 3.2 in the form used by the proof.

**Lemma A.1.** *For a per-step signal  $c : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ , let  $J_c(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} [\sum_{t \geq 0} \gamma^t c(s_t, a_t)]$ . Under the usual regularity conditions,*

$$\nabla_\theta J_c(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[ \sum_{t \geq 0} \gamma^t \nabla_\theta \log \pi_\theta(a_t | s_t) Q_c^{\pi_\theta}(s_t, a_t) \right],$$

and both  $Q_c^{\pi_\theta}$  and  $\nabla_\theta J_c$  are linear in  $c$ .

The identity is the policy-gradient theorem (Williams, 1992) with  $c$  in place of the task reward, and linearity follows from linearity of expectation. By the definitions in (2) and (3),  $f(\theta) = J_r(\theta)$  and  $\mathbf{v}^\top \mathbf{b}(\theta) = J_{\mathbf{v}^\top \mathbf{m}}(\theta)$  for any fixed  $\mathbf{v} \in \mathbb{R}^d$ .

*Proof of Proposition 4.1.* Write  $e_{ki} = \exp(-\|\mathbf{b}_k - \mathbf{b}_i\|_2^2 / \ell^2)$ . Only the terms of (1) that contain index  $k$  depend on  $\theta_k$ , so

$$\nabla_{\theta_k} S(\theta) = \nabla_{\theta_k} f_k - \sum_{i \neq k} \nabla_{\theta_k} [\sqrt{f_k f_i} e_{ki}].$$

Since  $f_k > 0$  and the policies are parameterized independently, the product and chain rules give

$$\nabla_{\theta_k} [\sqrt{f_k f_i} e_{ki}] = \frac{1}{2} \sqrt{\frac{f_i}{f_k}} e_{ki} \nabla_{\theta_k} f_k - \frac{2\sqrt{f_k f_i}}{\ell^2} e_{ki} (\nabla_{\theta_k} \mathbf{b}_k)^\top (\mathbf{b}_k - \mathbf{b}_i),$$

where  $\nabla_{\theta_k} \mathbf{b}_k$  is the Jacobian of  $\mathbf{b}(\theta_k)$ . Summing over  $i \neq k$  and collecting terms yields (7),

$$\nabla_{\theta_k} S(\theta) = W_k \nabla_{\theta_k} f_k + (\nabla_{\theta_k} \mathbf{b}_k)^\top \mathbf{c}_k,$$

with  $W_k$  and  $\mathbf{c}_k$  given by (8) and (9). Both are evaluated at the current population, so they are fixed numbers in this expression.

It remains to express both terms as policy gradients. The first is  $W_k \nabla_{\theta_k} J_r(\theta_k) = \nabla_{\theta_k} J_{W_k r}(\theta_k)$ . For the second,  $(\nabla_{\theta_k} \mathbf{b}_k)^\top \mathbf{c}_k = \nabla_{\theta_k} [\mathbf{v}^\top \mathbf{b}(\theta_k)]_{\mathbf{v}=\mathbf{c}_k} = \nabla_{\theta_k} J_{\mathbf{c}_k^\top \mathbf{m}}(\theta_k)$  with  $\mathbf{c}_k$  held fixed. By the linearity in Lemma A.1, the sum of the two gradients is the policy gradient of  $J_{\tilde{r}_k}$  for  $\tilde{r}_k = W_k r + \mathbf{c}_k^\top \mathbf{m}$ , which is (10).  $\square$

Note that holding  $W_k$  and  $\mathbf{c}_k$  fixed is not a stop-gradient approximation. The chain rule has already accounted for their dependence on  $\theta_k$ , which is what produces the second term of  $W_k$  and the direction  $\mathbf{c}_k$ .

### A.2. From the objective to the RePPO update

**Interactions in transformed coordinates.** Descriptors in our domains are normalized to  $[-1, 1]^d$ , while the Gaussian kernel treats behavior space as unbounded. Following SQUAD (Hedayatian & Nikolaidis, 2026b), we therefore define the interaction on the logit coordinates  $g(\mathbf{b})$  of Section 4.2, replacing  $e_{ki}$  with  $\exp(-\|g(\mathbf{b}_k) - g(\mathbf{b}_i)\|_2^2 / \ell^2)$ . Repeating the proof with this kernel leaves  $W_k$  unchanged in form and replaces the behavior term by  $(\nabla_{\theta_k} \mathbf{b}_k)^\top J_g(\mathbf{b}_k)^\top \mathbf{c}_k^g$ , where  $\mathbf{c}_k^g$  is (9) evaluated at the transformed descriptors and

$$J_g(\mathbf{b}) = \text{diag} \left( \frac{2}{1 - b_1^2}, \dots, \frac{2}{1 - b_d^2} \right) \quad (13)$$

is the Jacobian of the componentwise logit  $g(b) = \log((1+b)/(1-b))$ . The shaped reward thus weights the original signals  $\mathbf{m}$  by  $J_g(\mathbf{b}_k)^\top \mathbf{c}_k^g$ , which is the conversion in (11). Because the Jacobian grows near the bounds, a descriptor that is close to saturation receives a larger weight, matching the stretching of the kernel geometry there. For numerical stability, the normalized coordinates  $(\hat{\mathbf{d}}_k + \mathbf{1})/2$  are clipped to  $[\epsilon_b, 1 - \epsilon_b]$  before both the logit and its Jacobian are evaluated.

**Departures from the exact gradient.** RePPO modifies this exact update in several ways to obtain a working algorithm. Here, we enumerate them while Appendix D examines the task coefficient, the cap, and the local interactions empirically.

1. *Smoothed episode statistics.* The qualities and descriptors are replaced by exponential moving averages of completed-episode returns and episode-level descriptors, which reduces noise in the interaction at the cost of a small lag. The averages are updated once per iteration from the episodes completed in that iteration, and a policy that completes no episode keeps its previous values. Following standard practice, the advantages use discounted GAE as a surrogate for the gradient of these undiscounted statistics.
2. *Shifted qualities.* The objective requires  $f_k > 0$ , whereas returns can be negative. RePPO uses  $f_k^+ = \hat{F}_k - \min_j \hat{F}_j + \epsilon_f$ , which is positive and preserves the ordering and differences of the population’s qualities.
3. *Local interactions.* The sum in (9) is restricted to the  $K$  nearest neighbors in transformed space. Under the Gaussian kernel, the omitted terms decay exponentially with distance, so the contribution of farther neighbors is naturally vanishing.
4. *Unit task coefficient and capped repulsion.* RePPO fixes  $W_k = 1$  and rescales the diversity weights to norm at most  $C_{\max}$ , preserving their direction.
5. *Channel standardization.* Each advantage channel is standardized before the channels are combined with (12). The diversity weights  $\mathbf{p}_k$  then set the relative influence of the channels on the update independently of their numerical scales.

## B. Evaluation protocol

**Re-evaluation.** After training, every policy is evaluated anew, and these evaluations define all reported numbers. In continuous control, each policy runs 256 episodes with deterministic actions. In Atari, each policy plays 32 full games, over all lives and up to the cap of 27000 agent steps. Finally, LLM members are evaluated on 8 samples per test problem. Fitness is the mean undiscounted return and the behavior descriptor is the mean episode descriptor over these evaluations.

**Quality filter.** For each task, let  $f^*$  be the fitness of the best policy found by any method in any seed. Each population or archive retains the policies whose fitness is at least  $0.1f^*$  ( $0.25f^*$  on Ant), and all metrics except best fitness are computed on the retained policies. The filter is applied identically to every method, and archives are otherwise used as they are. Note that in the LLM experiments, we remove no members.

**Metrics.** For retained policies with mean descriptors  $\bar{\mathbf{b}}_1, \dots, \bar{\mathbf{b}}_n \in [-1, 1]^d$ , let  $G_{ij} = \exp(-\|\bar{\mathbf{b}}_i - \bar{\mathbf{b}}_j\|_2^2 / (2\sigma^2))$  with  $\sigma^2 = 0.5$ , and let  $\lambda_1, \dots, \lambda_n$  be the eigenvalues of  $G/n$ . The Vendi Score (Friedman & Dieng, 2023) is  $\exp(-\sum_i \lambda_i \log \lambda_i)$ , which ranges from 1, when all descriptors coincide, to  $n$ , when they are mutually far apart. The quality-weighted Vendi Score (Nguyen & Dieng, 2024) is the mean fitness of the retained policies multiplied by their Vendi Score. In continuous control and Atari, we report means and standard errors over seeds. In the LLM experiments, fitness is a member’s pass@1, the descriptors are averaged over the member’s test solutions.

## C. Continuous control

### C.1. Tasks and descriptors

We use the Ant, HalfCheetah, Humanoid, and Walker2D tasks of Brax 0.14 (Freeman et al., 2021) with the MJX physics backend and episodes of at most 1000 steps. Fitness is the default task reward, summed over an episode. The descriptor is the fraction of the episode each foot spends in contact, mapped to  $[-1, 1]$ . The tracked links are the four lower legs of Ant, the two feet of HalfCheetah and Walker2D, and Humanoid. During training, the behavior signal of a link is its signed contact indicator  $2c_t - 1$ .

### C.2. RePPO and Ind-PPO

Each policy has separate actor, task-critic, and behavior-critic networks, each an MLP with two hidden layers of 128 tanh units. The actor outputs the mean of a diagonal Gaussian with a state-independent standard deviation, and actions are

squashed with  $\tanh$ . Observations are normalized with per-policy running statistics and clipped to  $\pm 10$ , and training rewards are clipped to  $\pm 10$  per step. Table 2 lists the hyperparameters. Ind-PPO uses identical settings with the diversity weights set to zero.

Table 2. RePPO hyperparameters in continuous control. The learning rate decays as  $\eta_t = \eta_0 \max(0.05, 1 - 2t/S)$  in the number of environment steps  $t$ .

|                                                    | Ant                | HalfCheetah | Humanoid  | Walker2D  |
|----------------------------------------------------|--------------------|-------------|-----------|-----------|
| Learning rate $\eta_0$                             | $3 \times 10^{-4}$ | $10^{-3}$   | $10^{-3}$ | $10^{-3}$ |
| Schedule horizon $S$                               | 2.05 B             | 4.10 B      | 4.10 B    | 4.10 B    |
| Update epochs                                      | 10                 | 10          | 10        | 5         |
| Entropy coefficient                                | 0.005              | 0           | 0         | 0         |
| Value clipping                                     | yes                | no          | no        | no        |
| Kernel bandwidth $\ell^2$                          | 0.1                | 0.5         | 0.1       | 0.1       |
| Shared across tasks                                |                    |             |           |           |
| Population size $N$                                | 32                 |             |           |           |
| Environments per policy / rollout length           | 64 / 1024          |             |           |           |
| Total environment steps                            | $2.05 \times 10^9$ |             |           |           |
| Minibatches                                        | 32                 |             |           |           |
| Discount $\gamma$ / GAE $\lambda$                  | 0.99 / 0.95        |             |           |           |
| PPO clip / value coefficient / max gradient norm   | 0.2 / 0.5 / 0.5    |             |           |           |
| Neighbors $K$ / repulsion cap $C_{\max}$           | 8 / 1.0            |             |           |           |
| EMA rates for $\hat{F}_k$ and $\hat{\mathbf{d}}_k$ | 0.99               |             |           |           |
| $\epsilon_f$ / $\epsilon_b$                        | $1 / 10^{-6}$      |             |           |           |

### C.3. Baselines

All baselines use the same tasks, descriptors, and evaluation as RePPO, and their archives are re-evaluated with the protocol of Appendix B. Table 3 summarizes their main settings.

**PPGA.** We port the official implementation (Batra et al., 2024) to the current version of Brax with the MJX backend. All hyperparameters follow the original work, except the number of outer iterations, which we set to 200 so that the budget approaches that of the other methods.

More specifically, PPGA uses one CMA-MAEGA emitter with 300 branching solutions, 10 gradient and 10 mean-update PPO iterations per outer iteration on 3000 parallel environments with rollouts of 128 steps, 4 epochs over 8 minibatches, value coefficient 2, no entropy bonus, and a target KL of 0.008. It uses archive learning rates of 0.1 (Ant, Humanoid) and 0.5 (HalfCheetah, Walker2D). The initial step sizes are 3.0, 1.0, 0.5, and 1.0 for Ant, HalfCheetah, Humanoid, and Walker2D, respectively. PPO’s learning rate is  $10^{-3}$ , except on Humanoid where it is  $3 \times 10^{-4}$ .

**DCRL-ME and PGA-ME.** Both build on the QDax implementations (Chalumeau et al., 2024) and use the hyperparameters reported by their authors, with the number of iterations matched to the budget.

In DCRL-ME each iteration produces 128 offspring by isoline variation ( $\sigma_{\text{iso}} = 0.005$ ,  $\sigma_{\text{line}} = 0.05$ ), 64 by 150 descriptor-conditioned policy-gradient steps (learning rate  $5 \times 10^{-3}$ ), and 64 by specializing the descriptor-conditioned actor. The critic has two hidden layers of 256 units and is trained for 3000 steps per iteration with TD3 settings (batch 100, replay buffer  $10^6$ , discount 0.99, policy noise 0.2, noise clip 0.5, target rate 0.005, policy delay 2).

PGA-ME generates 300 offspring in each iteration, half by isoline variation and half by 100 policy-gradient steps, with 300 critic updates per iteration.

**Sep-CMA-MAE.** We use the pyribs implementation (Tjanaka et al., 2023b) with the hyperparameters of Tjanaka et al. (2023a), evaluating each candidate on one episode. In particular, it has five emitters with batches of 40 using an initial step size of 0.02, an archive learning rate of 0.001, improvement ranking, and the basic restart rule. Its budget is 2.048 M evaluations of maximum length 1000.

Table 3. Main settings of the archive-based baselines in continuous control. Environment steps count full-length evaluation episodes, except for Sep-CMA-MAE, whose budget is a number of evaluations.

|                    | PPGA                       | DCRL-ME                | PGA-ME                     | Sep-CMA-MAE               |
|--------------------|----------------------------|------------------------|----------------------------|---------------------------|
| Archive            | grid, $50^2$ ( $10^4$ Ant) | CVT, 1024              | grid, $50^2$ ( $10^4$ Ant) | grid, $32^2$ ( $6^4$ Ant) |
| Iterations         | 200                        | 8000                   | 6827                       | –                         |
| Policy network     | 128-128 tanh               | 128-128 ReLU           | 128-128 ReLU               | 128-128 tanh              |
| Gradient estimator | PPO, 3000 envs             | TD3, desc.-conditioned | TD3                        | –                         |
| Environment steps  | 2.74 B                     | 2.05 B                 | 2.05 B                     | $\leq 2.05$ B             |

#### C.4. Additional results

Table 4 reports the numbers behind the top row of Figure 1, together with the best fitness and the number of policies that pass the quality filter. Despite the larger population sizes of archive-based methods, RePPO has the highest Vendi Score on two of the tasks and the second highest on the other two.

Table 4. Final populations in continuous control, as mean  $\pm$  standard error over seeds. All other metrics belong to the retained policies

| Task        | Method      | Best fitness                     | Mean fitness                    | Vendi Score                       | qVS                               | Retained |
|-------------|-------------|----------------------------------|---------------------------------|-----------------------------------|-----------------------------------|----------|
| Ant         | RePPO       | 4090 $\pm$ 68                    | 3011 $\pm$ 68                   | 1.82 $\pm$ 0.09                   | <b>5492 <math>\pm</math> 328</b>  | 12       |
|             | DCRL-ME     | 1335 $\pm$ 48                    | 1174 $\pm$ 10                   | <b>2.69 <math>\pm</math> 0.21</b> | 3157 $\pm$ 258                    | 61       |
|             | PPGA        | 1183 $\pm$ 113                   | 752 $\pm$ 309                   | 1.16 $\pm$ 0.48                   | 1439 $\pm$ 588                    | 2        |
|             | PGA-ME      | 786 $\pm$ 32                     | 0 $\pm$ 0                       | 0.00 $\pm$ 0.00                   | 0 $\pm$ 0                         | 0        |
|             | Sep-CMA-MAE | 306 $\pm$ 175                    | 0 $\pm$ 0                       | 0.00 $\pm$ 0.00                   | 0 $\pm$ 0                         | 0        |
|             | Ind-PPO     | <b>4339 <math>\pm</math> 31</b>  | <b>3683 <math>\pm</math> 72</b> | 1.10 $\pm$ 0.05                   | 4023 $\pm$ 103                    | 18       |
| HalfCheetah | RePPO       | 9536 $\pm$ 274                   | 4419 $\pm$ 82                   | <b>6.02 <math>\pm</math> 0.12</b> | <b>26571 <math>\pm</math> 263</b> | 30       |
|             | DCRL-ME     | 7449 $\pm$ 234                   | 4471 $\pm$ 154                  | 5.55 $\pm$ 0.03                   | 24801 $\pm$ 847                   | 1024     |
|             | PPGA        | 5398 $\pm$ 906                   | 2659 $\pm$ 390                  | 3.17 $\pm$ 0.43                   | 8104 $\pm$ 1217                   | 696      |
|             | PGA-ME      | 5798 $\pm$ 147                   | 1972 $\pm$ 90                   | 3.78 $\pm$ 0.07                   | 7433 $\pm$ 237                    | 1285     |
|             | Sep-CMA-MAE | 4677 $\pm$ 275                   | 3005 $\pm$ 175                  | 2.71 $\pm$ 0.13                   | 8197 $\pm$ 778                    | 402      |
|             | Ind-PPO     | <b>10185 <math>\pm</math> 29</b> | <b>9125 <math>\pm</math> 58</b> | 1.04 $\pm$ 0.00                   | 9454 $\pm$ 72                     | 32       |
| Humanoid    | RePPO       | 8190 $\pm$ 134                   | 6130 $\pm$ 64                   | <b>6.38 <math>\pm</math> 0.09</b> | <b>39089 <math>\pm</math> 286</b> | 32       |
|             | DCRL-ME     | 7594 $\pm$ 456                   | 4263 $\pm$ 125                  | 3.27 $\pm$ 0.08                   | 13964 $\pm$ 672                   | 749      |
|             | PPGA        | <b>9907 <math>\pm</math> 303</b> | 6298 $\pm$ 115                  | 2.99 $\pm$ 0.38                   | 18792 $\pm$ 2322                  | 598      |
|             | PGA-ME      | 4961 $\pm$ 121                   | 2026 $\pm$ 60                   | 1.43 $\pm$ 0.05                   | 2918 $\pm$ 191                    | 720      |
|             | Sep-CMA-MAE | 576 $\pm$ 8                      | 0 $\pm$ 0                       | 0.00 $\pm$ 0.00                   | 0 $\pm$ 0                         | 0        |
|             | Ind-PPO     | 8601 $\pm$ 134                   | <b>6744 <math>\pm</math> 80</b> | 1.28 $\pm$ 0.01                   | 8654 $\pm$ 147                    | 32       |
| Walker2D    | RePPO       | <b>7764 <math>\pm</math> 274</b> | 4035 $\pm$ 60                   | 4.59 $\pm$ 0.14                   | <b>18511 <math>\pm</math> 598</b> | 31       |
|             | DCRL-ME     | 6512 $\pm$ 172                   | 3214 $\pm$ 83                   | <b>5.21 <math>\pm</math> 0.06</b> | 16769 $\pm$ 593                   | 978      |
|             | PPGA        | 6866 $\pm$ 513                   | 2608 $\pm$ 549                  | 4.26 $\pm$ 0.43                   | 10426 $\pm$ 1283                  | 1196     |
|             | PGA-ME      | 3255 $\pm$ 156                   | 1391 $\pm$ 71                   | 3.63 $\pm$ 0.14                   | 5047 $\pm$ 343                    | 1156     |
|             | Sep-CMA-MAE | 3365 $\pm$ 122                   | 1653 $\pm$ 51                   | 4.10 $\pm$ 0.08                   | 6792 $\pm$ 309                    | 925      |
|             | Ind-PPO     | 7624 $\pm$ 276                   | <b>6191 <math>\pm</math> 83</b> | 1.15 $\pm$ 0.02                   | 7143 $\pm$ 161                    | 32       |

**Effect of the quality filter.** Diversity metrics can be dominated by policies that behave differently but do not solve the task. Figure 4 therefore recomputes the Vendi Score and qVS of the complete populations and archives while the required fraction of  $f^*$  increases from 0 to 0.5. With no filter, the archives of DCRL-ME and PPGA are far more diverse than RePPO’s population on Ant, but their Vendi Scores collapse as the threshold is increased. Upon manual inspection, we noticed that many of the policies in archive-based methods stand nearly still and occupy different cells by holding different subsets of legs off the ground, which prompted our choice of a 25% threshold for this task. Beyond this threshold, RePPO attains the highest qVS.

**Populations in behavior space.** Figure 5 shows the retained policies of one seed per method in tasks with two-dimensional behavior spaces, and Figure 6 shows the distribution of their fitness. Archive-based methods cover behavior space more uniformly due to their structure, but much of that coverage comes from low-fitness policies. RePPO places its policies less uniformly and concentrates them in regions that support fast locomotion, as the Soft QD objective rewards illuminating

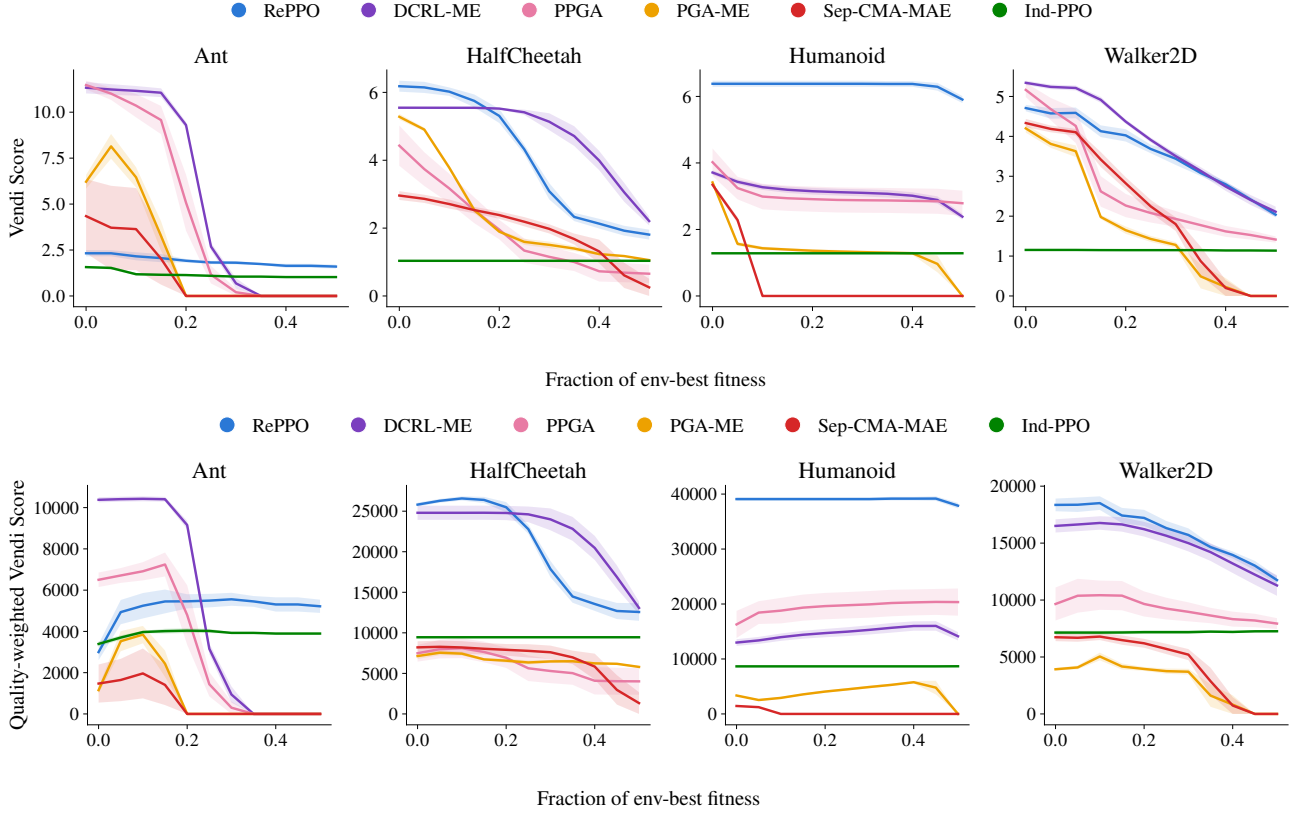


Figure 4. Vendi Score (top) and qVS (bottom) of the complete populations as a function of the minimum fitness, given as a fraction of the best policy found on the task. Lines show the mean over seeds and shaded regions one standard error.

behavior space with high-quality policies rather than covering it uniformly. Independent PPO concentrates all the policies in a small, high-quality region.

**Behaviors.** Figures 7–9 show gaits found by a single RePPO population in each task, which differ in which feet carry the body and how often they touch the ground.

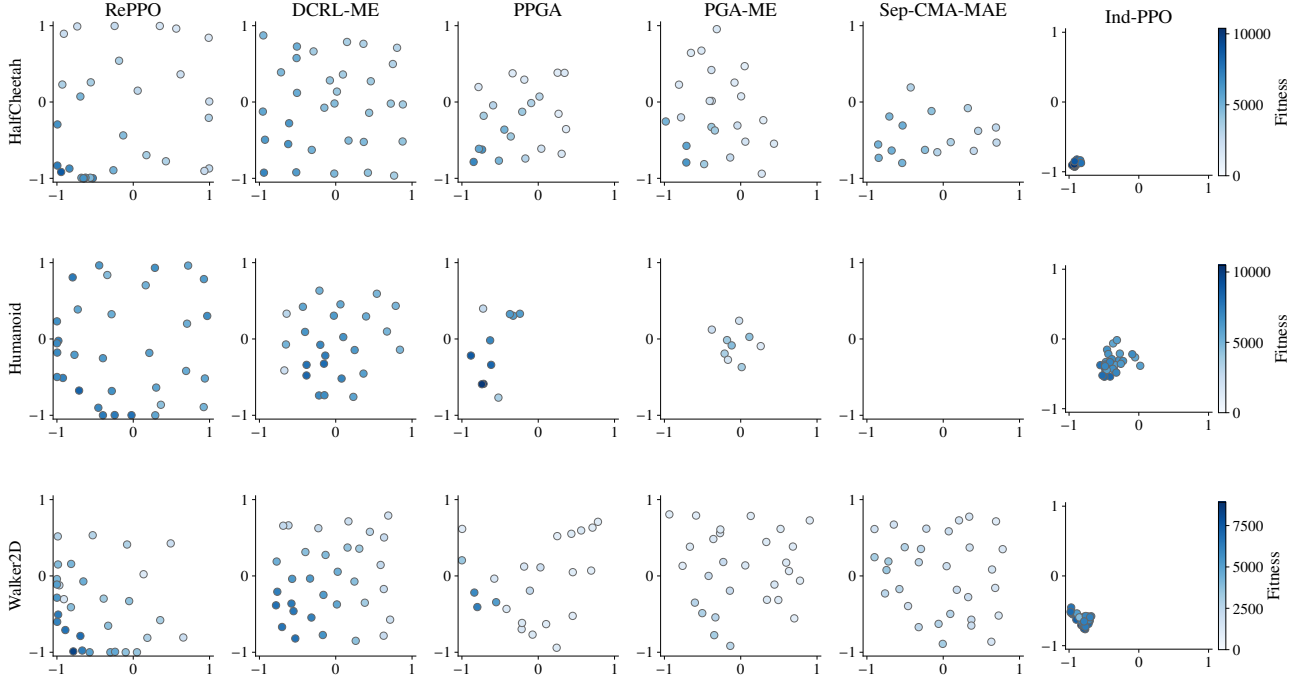


Figure 5. Retained policies of one seed per method in behavior space, colored by fitness on a shared scale within each task. For readability, archives are thinned to one policy in each of 32 CVT cells, keeping the fittest.

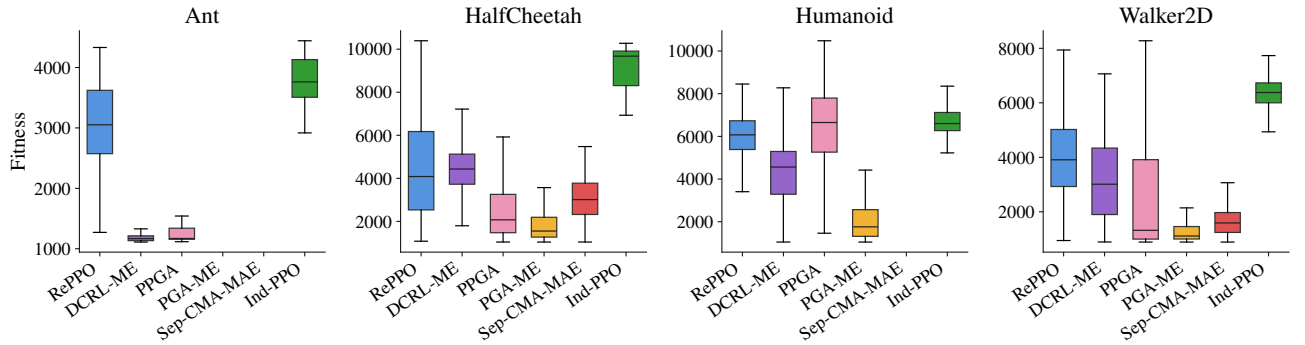


Figure 6. Fitness of the retained policies, pooled over seeds. Boxes show the interquartile range and medians.

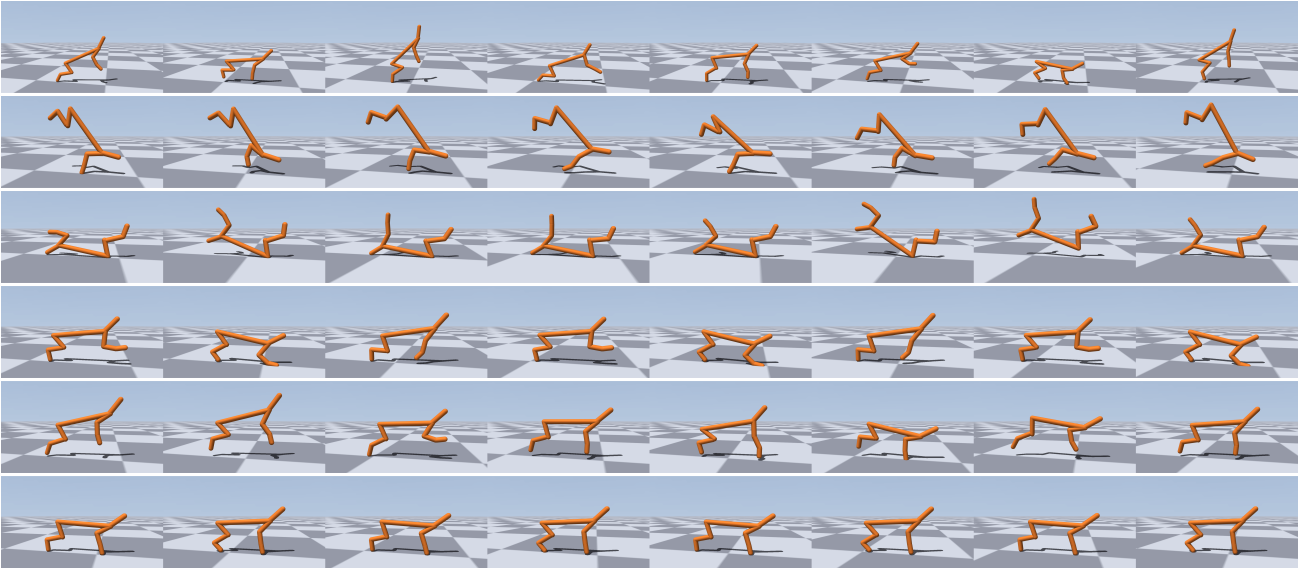


Figure 7. Gaits found by one RePPO population in HalfCheetah.

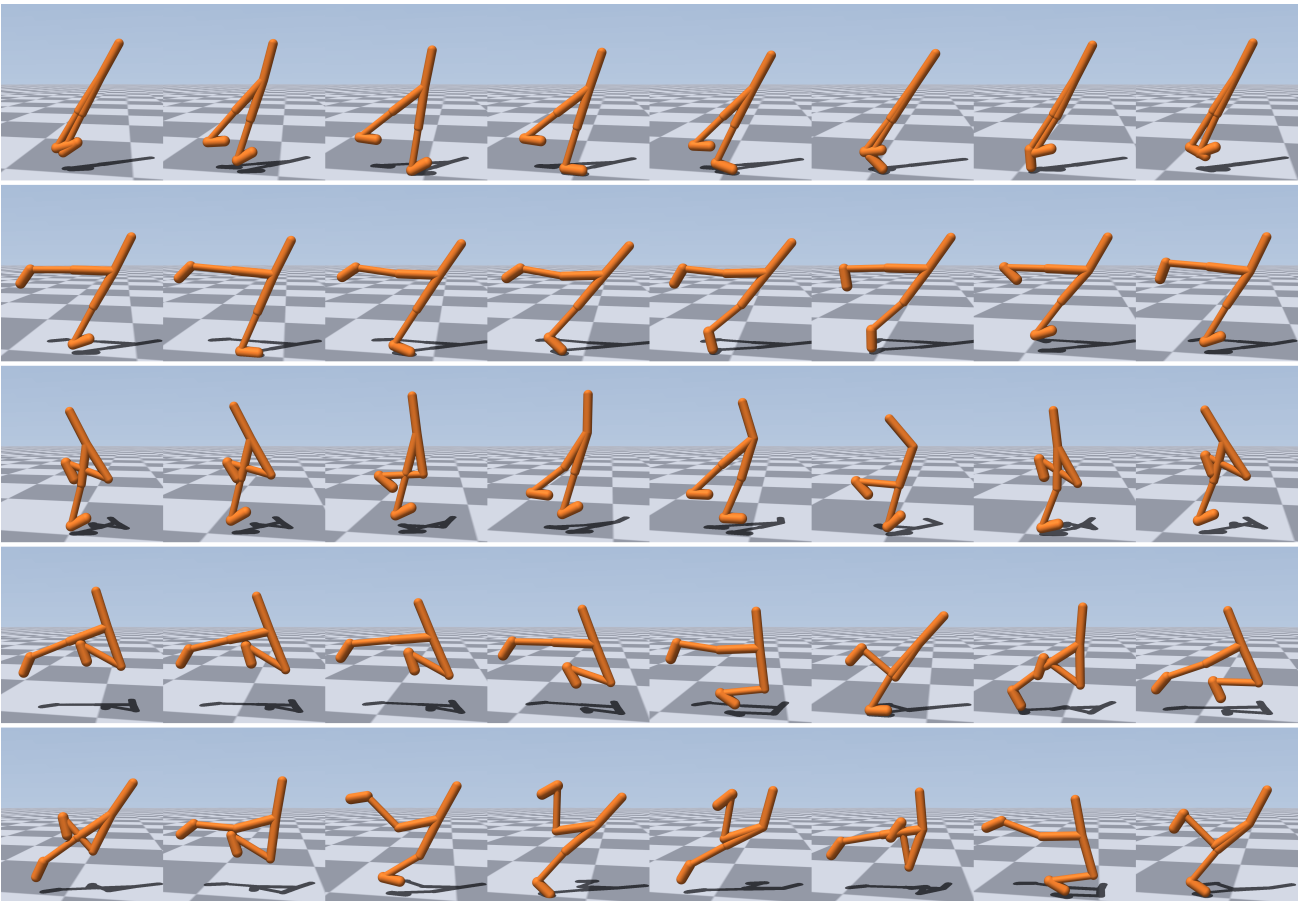


Figure 8. Gaits found by one RePPO population in Walker2D.

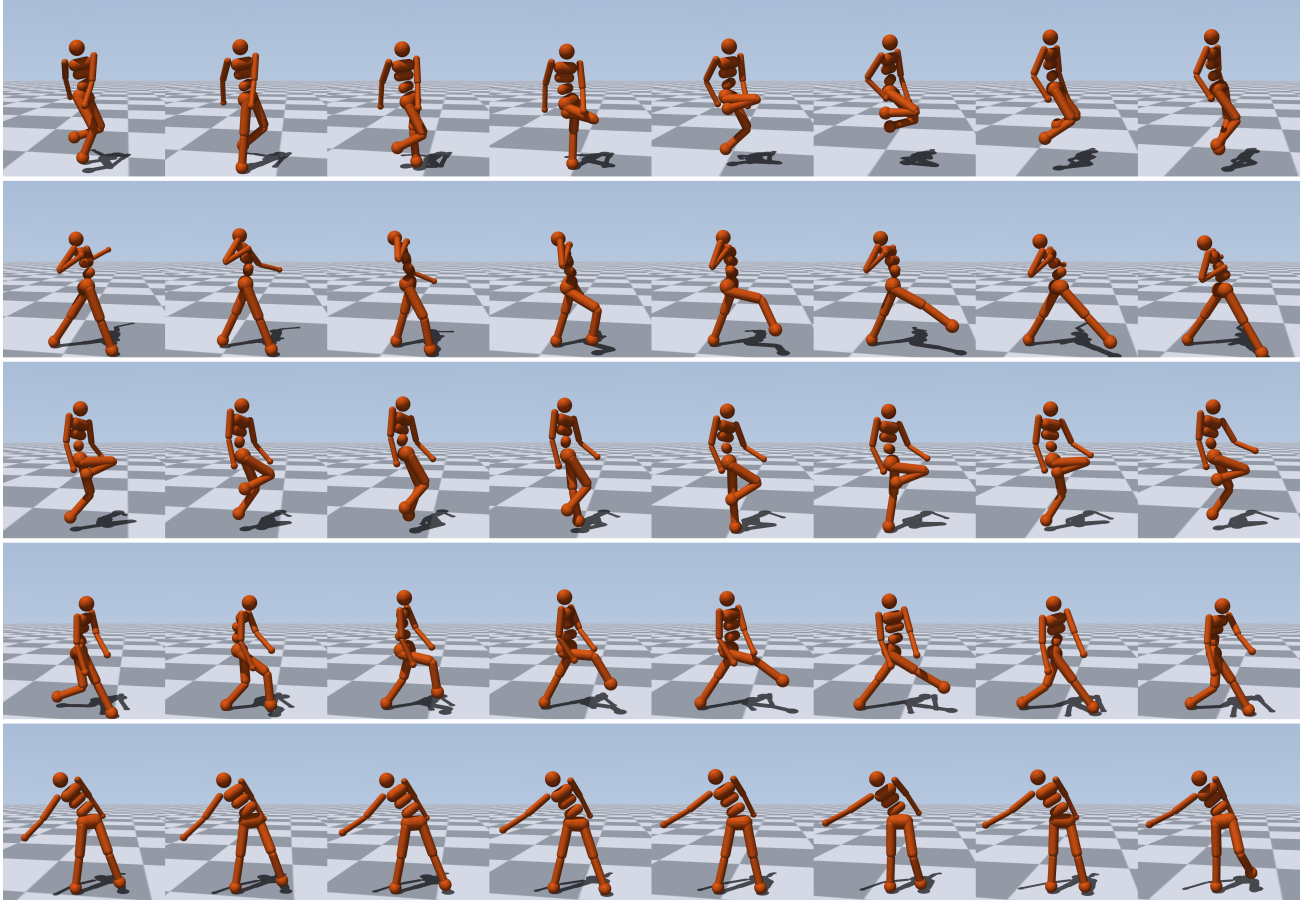


Figure 9. Gaits found by one RePPO population in Humanoid.

## D. Ablations

We ablate RePPO’s components and hyperparameters on Humanoid. Each variant changes one setting of the main Humanoid configuration ( $C_{\max} = 1$ ,  $\ell^2 = 0.1$ ,  $K = 8$ ,  $N = 32$ , symmetric repulsion,  $W_k = 1$ ) and runs for three seeds. The default is represented by the first three seeds of the main runs. All metrics follow Appendix B, with the quality threshold of the main comparison. We additionally report the spread of a population, the mean Euclidean distance of its policies from their centroid in the normalized behavior space  $[-1, 1]^d$ , which characterizes its geometry alongside the Vendi Score. Table 5 lists every variant.

### D.1. Structural choices

We first examine two structural choices: the unit task coefficient of Section 4.1, and symmetric interactions, in which every pair of neighbors repels each other regardless of their qualities.

**Exact task coefficient.** Using the exact coefficient  $W_k$  of (8) causes training to collapse in every seed. Policies perform significantly worse with the best policy reaching a fitness of only  $282 \pm 230$ . When several high-quality policies share a region of behavior space—which happens often early in training—the second term of  $W_k$  outweighs the first and the task coefficient becomes negative. The policy is then rewarded for lowering its return, and once it reaches low-return behaviors, such as falling immediately, PPO does not recover. Fixing  $W_k = 1$  preserves a consistent task-improvement signal and leaves behavioral separation to the descriptor term.

**Asymmetric repulsion.** In this variant, a policy is repelled only by neighbors of higher quality, so that the best policies are never displaced by weaker ones. The best policy is unaffected ( $8409 \pm 201$  versus  $8336 \pm 68$ ), but the rest of the population is both less fit and less diverse, lowering qVS from 39212 to 34086. Symmetric interactions let every overlapping pair separate, which spreads the population more evenly.

### D.2. Interaction strength and scale

Together, the repulsion cap  $C_{\max}$  and the kernel bandwidth  $\ell^2$  shape how the population spreads through behavior space, but they do so in different ways (Figures 10 and 11).

**Repulsion cap.** The cap controls how strongly diversity enters each update, with  $C_{\max} = 0$  recovering Ind-PPO. Raising it from 0.1 to 1 increases the Vendi Score from 3.92 to 6.33 and the spread from 0.60 to 0.91, while mean fitness stays between 5700 and 6200. Beyond  $C_{\max} = 2$ , diversity saturates and the stronger diversity signal begins to compete with task learning: at  $C_{\max} = 8$ , mean and best fitness fall to 4691 and 6817. qVS peaks at the default of 1, with  $C_{\max} = 2$  close behind. The cap therefore mainly controls how far the population expands, and needs to be large enough to spread the policies but not so large that repulsion dominates the update.

**Kernel bandwidth.** The bandwidth sets the scale over which policies interact and thus the configuration the population settles into. With  $\ell^2 = 0.02$ , interactions are highly local, and the population remains compact (spread 0.70, Vendi Score 4.80) and fit (mean fitness 6703). Increasing the bandwidth pushes policies apart until the spread saturates near 1.0 at  $\ell^2 = 0.5$ , where the Vendi Score peaks at 6.84. Larger bandwidths push policies toward the corners of behavior space, leaving its center empty, so that diversity declines slightly while mean fitness stays near 5200. qVS peaks at  $\ell^2 = 0.1$ , which balances these effects on Humanoid.

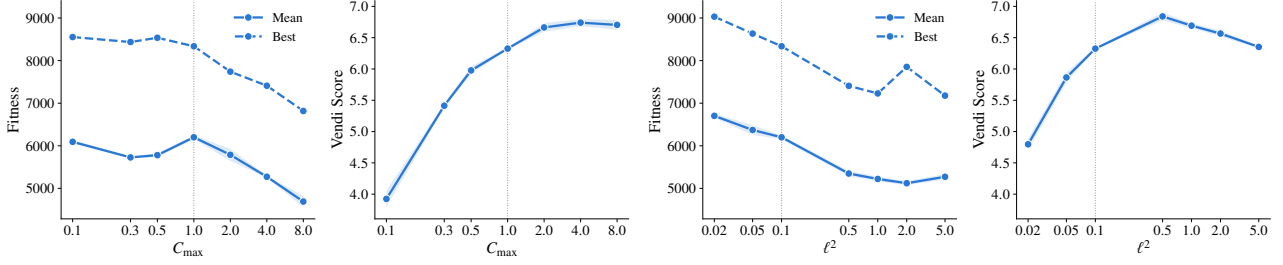


Figure 10. Effect of the repulsion cap  $C_{\max}$  (left) and the kernel bandwidth  $\ell^2$  (right) on Humanoid. Shaded regions show one standard error over seeds.

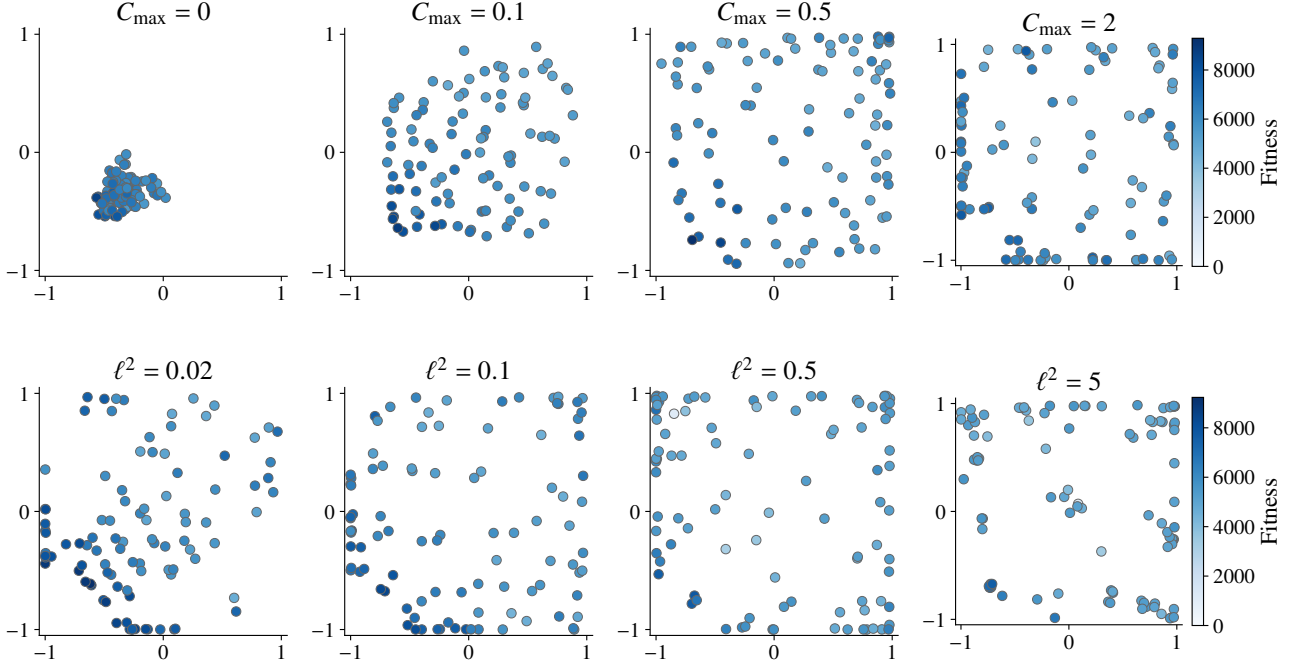


Figure 11. Final Humanoid populations for representative values of  $C_{\max}$  (top) and  $\ell^2$  (bottom), pooled over seeds and colored by fitness.  $C_{\max} = 0$  corresponds to Ind-PPO.

### D.3. Local interactions

Restricting the interactions to  $K$  nearest neighbors has little effect over the range we tested (Figure 12, left). With  $K \in \{4, 8, 16\}$ , the Vendi Score lies between 6.33 and 6.50 and qVS within 4% of the default. Interacting with the entire population ( $K = 31$ ) produces slightly fitter but less diverse populations. We therefore conclude that the nearest neighbors carry most of the interaction, and  $K = 8$  retains its effect at lower cost.

### D.4. Population size at a fixed budget

We vary the population size while keeping the total number of environment steps fixed, so that larger populations give each policy less experience (Figure 12, right). Diversity grows with population size with diminishing returns, from a Vendi Score of 3.76 at  $N = 8$  to 6.97 at  $N = 64$ . Mean fitness is highest for  $N = 16$  and  $N = 32$  and drops at  $N = 64$ , where each policy receives half the experience of the default, whereas the best fitness remains above 7200 for every size. A larger population therefore represents behavior space more densely until the budget is spread too thinly, and  $N = 32$  balances these effects.

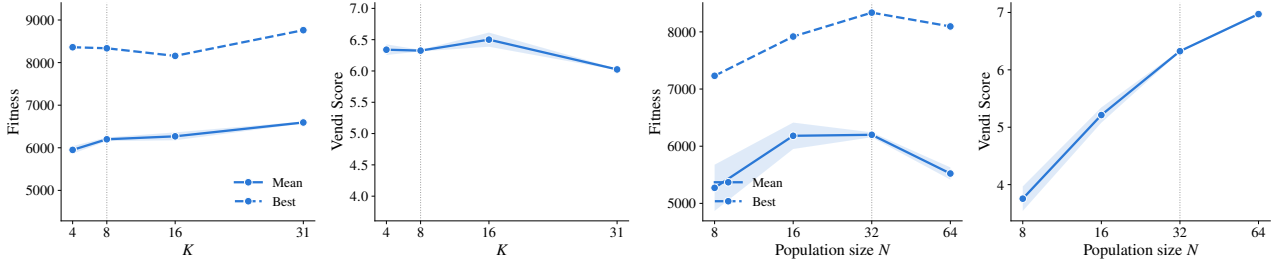


Figure 12. Effect of the number of neighbors  $K$  (left) and of the population size  $N$  at a fixed budget (right) on Humanoid. Shaded regions show one standard error over seeds.

Table 5. All Humanoid ablations, as mean  $\pm$  standard error over three seeds. The default configuration appears in bold.

|            | Setting              | Mean fitness                    | Best fitness                    | Vendi Score                       | qVS                               | Spread      |
|------------|----------------------|---------------------------------|---------------------------------|-----------------------------------|-----------------------------------|-------------|
|            | <b>Default</b>       | <b>6200 <math>\pm</math> 43</b> | <b>8336 <math>\pm</math> 68</b> | <b>6.33 <math>\pm</math> 0.01</b> | <b>39212 <math>\pm</math> 231</b> | <b>0.91</b> |
| Structure  | Asymmetric repulsion | 5935 $\pm$ 144                  | 8409 $\pm$ 201                  | 5.74 $\pm$ 0.21                   | 34086 $\pm$ 1688                  | 0.83        |
|            | Exact $W_k$          | 0 $\pm$ 0                       | 282 $\pm$ 230                   | 0.00 $\pm$ 0.00                   | 0 $\pm$ 0                         | –           |
| $C_{\max}$ | 0.1                  | 6093 $\pm$ 42                   | 8553 $\pm$ 138                  | 3.92 $\pm$ 0.13                   | 23901 $\pm$ 681                   | 0.60        |
|            | 0.3                  | 5727 $\pm$ 18                   | 8436 $\pm$ 195                  | 5.41 $\pm$ 0.03                   | 31003 $\pm$ 282                   | 0.79        |
|            | 0.5                  | 5780 $\pm$ 47                   | 8536 $\pm$ 443                  | 5.98 $\pm$ 0.06                   | 34544 $\pm$ 88                    | 0.87        |
|            | 2.0                  | 5788 $\pm$ 143                  | 7740 $\pm$ 67                   | 6.66 $\pm$ 0.07                   | 38581 $\pm$ 1225                  | 0.94        |
|            | 4.0                  | 5273 $\pm$ 22                   | 7410 $\pm$ 75                   | 6.74 $\pm$ 0.06                   | 35542 $\pm$ 444                   | 0.95        |
|            | 8.0                  | 4691 $\pm$ 132                  | 6817 $\pm$ 51                   | 6.70 $\pm$ 0.08                   | 31466 $\pm$ 1253                  | 0.96        |
| $\ell^2$   | 0.02                 | 6703 $\pm$ 48                   | 9031 $\pm$ 100                  | 4.80 $\pm$ 0.09                   | 32149 $\pm$ 551                   | 0.70        |
|            | 0.05                 | 6371 $\pm$ 114                  | 8633 $\pm$ 276                  | 5.86 $\pm$ 0.08                   | 37367 $\pm$ 952                   | 0.83        |
|            | 0.5                  | 5348 $\pm$ 60                   | 7407 $\pm$ 338                  | 6.84 $\pm$ 0.06                   | 36586 $\pm$ 712                   | 1.03        |
|            | 1.0                  | 5221 $\pm$ 59                   | 7229 $\pm$ 388                  | 6.69 $\pm$ 0.04                   | 34943 $\pm$ 583                   | 1.00        |
|            | 2.0                  | 5121 $\pm$ 45                   | 7852 $\pm$ 200                  | 6.57 $\pm$ 0.05                   | 33626 $\pm$ 509                   | 1.01        |
|            | 5.0                  | 5272 $\pm$ 67                   | 7177 $\pm$ 363                  | 6.35 $\pm$ 0.02                   | 33493 $\pm$ 539                   | 0.97        |
| $K$        | 4                    | 5950 $\pm$ 89                   | 8361 $\pm$ 203                  | 6.34 $\pm$ 0.08                   | 37724 $\pm$ 791                   | 0.91        |
|            | 16                   | 6269 $\pm$ 93                   | 8157 $\pm$ 92                   | 6.50 $\pm$ 0.11                   | 40734 $\pm$ 333                   | 0.92        |
|            | 31                   | 6594 $\pm$ 24                   | 8761 $\pm$ 196                  | 6.02 $\pm$ 0.01                   | 39725 $\pm$ 128                   | 0.90        |
| $N$        | 8                    | 5272 $\pm$ 404                  | 7231 $\pm$ 153                  | 3.76 $\pm$ 0.21                   | 19820 $\pm$ 1979                  | 0.65        |
|            | 16                   | 6182 $\pm$ 231                  | 7918 $\pm$ 512                  | 5.21 $\pm$ 0.13                   | 32183 $\pm$ 886                   | 0.79        |
|            | 64                   | 5522 $\pm$ 107                  | 8095 $\pm$ 227                  | 6.97 $\pm$ 0.03                   | 38492 $\pm$ 619                   | 1.00        |

## E. Atari

### E.1. Games and descriptors

We use the ALE v5 versions (Bellemare et al., 2015) of Boxing, Asterix, Seaquest, and Ms. Pac-Man through EnvPool (Weng et al., 2022), with the standard preprocessing: a frame skip of 4 with max-pooling over the last two frames, 0 to 30 no-op actions at reset, grayscale frames resized to  $84 \times 84$ , and a stack of the last 4 frames. Following Machado et al. (2018), the emulator repeats the previous action with probability 0.25 during both training and evaluation. A game lasts until all lives are lost or 27000 agent steps have passed. Fitness is the raw game score of a full game. During training, the task reward is clipped to its sign and a lost life ends the episode for the task critic.

Each game defines a two-dimensional behavior signal  $\mathbf{m}_t$  from the console memory, and the descriptor of a game is its average over all agent steps, clipped to  $[-1, 1]$ . Table 6 defines the signals and Figure 13 illustrates the games. For the three positional descriptors, the affine scales map the range that competent play reaches to roughly  $[-1, 1]$ . In Ms. Pac-Man, eating a dot after  $k$  of the level’s  $n$  dots have been eaten contributes  $w_k = 1 - 2k/(n - 1)$ , from +1 for the first dot to  $-1$  for the last, signed by the half of the maze in which the dot lies, so that the descriptor records which halves are cleared first and is zero when the maze is cleared uniformly.

Table 6. Behavior signals in Atari. Positional signals are affine in the memory values and clipped to  $[-1, 1]$  per step.

| Game        | First dimension ( $-1 \rightarrow +1$ )                         | Second dimension ( $-1 \rightarrow +1$ )                        |
|-------------|-----------------------------------------------------------------|-----------------------------------------------------------------|
| Boxing      | distance to the opponent (close $\rightarrow$ far)              | height in the ring (top $\rightarrow$ bottom)                   |
| Asterix     | horizontal position (left $\rightarrow$ right)                  | lane (top $\rightarrow$ bottom)                                 |
| Seaquest    | horizontal position (left $\rightarrow$ right)                  | depth (just below the surface $\rightarrow$ sea floor)          |
| Ms. Pac-Man | clearing order (left half first $\rightarrow$ right half first) | clearing order (top half first $\rightarrow$ bottom half first) |

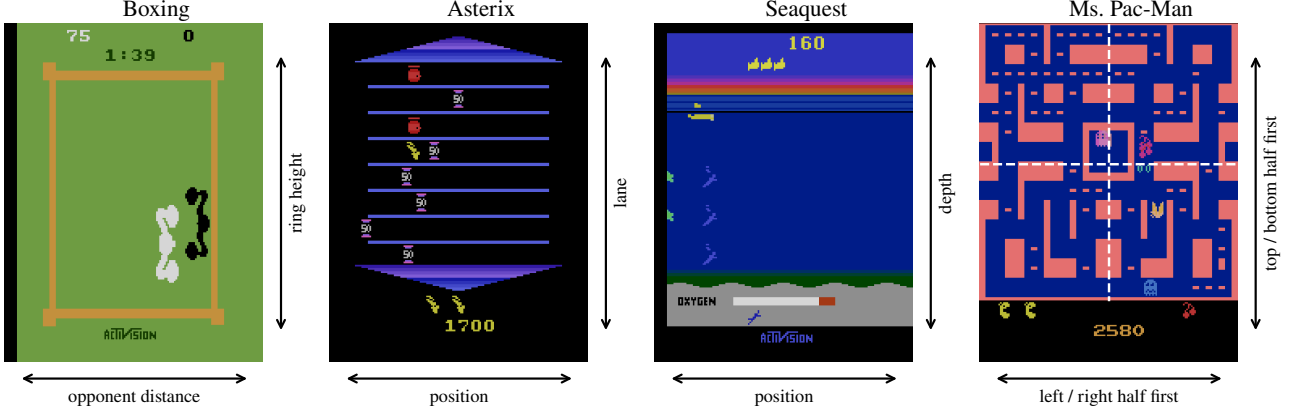


Figure 13. The four Atari games, as played by the best RePPO policy of one population, with the two behavior dimensions of each game indicated by arrows.

## E.2. RePPO and Ind-PPO

Each policy is a Nature CNN (Mnih et al., 2015) with linear actor, task-critic, and two-output behavior-critic heads on its 512-dimensional features, totaling 1.69 M parameters. The policies share no parameters. PPO settings follow the standard Atari configuration (Huang et al., 2022), and Table 7 lists all hyperparameters. Two settings adapt the behavior channels to full games of several thousand steps. First, the behavior channels end only when the game ends rather than at every lost life. Second, their rewards are scaled by  $1 - \gamma$  so that the behavior returns stay on the scale of the descriptor itself. The EMA rates are lower than in continuous control because a game spans many iterations, and the statistics are updated only in iterations in which a game ends.

Table 7. RePPO hyperparameters in Atari. Ind-PPO uses the same settings with the diversity weights set to zero.

| Parameter                   | Value                                    | Parameter                                | Value                    |
|-----------------------------|------------------------------------------|------------------------------------------|--------------------------|
| Population size $N$         | 8                                        | Discount $\gamma$ / GAE $\lambda$        | 0.99 / 0.95              |
| Environments per policy     | 8                                        | PPO clip                                 | 0.1, with value clipping |
| Rollout length              | 128                                      | Entropy / value coefficient              | 0.01 / 0.5               |
| Steps per policy            | 10 M                                     | Max gradient norm                        | 0.5                      |
| Learning rate               | $2.5 \times 10^{-4}$ , linear decay to 0 | Neighbors $K$ / repulsion cap $C_{\max}$ | 7 / 1.0                  |
| Update epochs / minibatches | 4 / 4                                    | Kernel bandwidth $\ell^2$                | 0.1                      |
| EMA rates                   | 0.9                                      | $\epsilon_f / \epsilon_b$                | 1 / $10^{-6}$            |

## E.3. RefQD

RefQD (Wang et al., 2024) reduces the cost of an archive of neural policies by splitting each network into a representation part, shared by the entire archive, and a decision part, the last layer, which is stored per cell. We use its Atari variant, Ref-DQN-ME, which produces offspring by isoline variation and by Double DQN updates (van Hasselt et al., 2016), with the authors' default hyperparameters: a CVT archive of 1000 cells that stores two decision parts per cell, 100 offspring per generation, 300 DQN training steps per generation, periodic re-evaluation of archived decision parts with the current representation, and a single grayscale frame as input. We set the budget to 80 M steps, which the implementation rounds up to 86.4 M so that every re-evaluation cycle completes. For Seaquest, we enlarge the replay buffer from  $2 \times 10^5$  to  $3 \times 10^5$ .

transitions due to longer episodes.

#### E.4. Additional results

Table 8 reports the numbers behind the bottom row of Figure 1. In Boxing, Ind-PPO’s policies keep the same distance from the opponent and mostly stay at one of two heights, whereas RePPO’s policies spread over the height of the ring and include a second family that fights at closer range. In Asterix and Ms. Pac-Man, Ind-PPO’s policies already vary along one dimension, horizontal position and left–right clearing order, which the symmetry of the games makes equally rewarding. RePPO additionally spreads its policies along the other dimension, covering the outer lanes in Asterix and including policies that clear the top half of the maze first in Ms. Pac-Man.

Table 8. Final populations in Atari, as mean  $\pm$  standard error over five seeds.

| Game        | Method  | Best score                       | Mean score                       | Vendi Score                       | qVS                               |
|-------------|---------|----------------------------------|----------------------------------|-----------------------------------|-----------------------------------|
| Boxing      | RePPO   | <b>98 <math>\pm</math> 0</b>     | 91 $\pm$ 1                       | <b>2.96 <math>\pm</math> 0.09</b> | <b>271 <math>\pm</math> 8</b>     |
|             | RefQD   | 33 $\pm$ 5                       | 17 $\pm$ 2                       | 1.88 $\pm$ 0.12                   | 32 $\pm$ 4                        |
|             | Ind-PPO | 96 $\pm$ 1                       | <b>93 <math>\pm</math> 0</b>     | 1.72 $\pm$ 0.18                   | 159 $\pm$ 16                      |
| Asterix     | RePPO   | <b>3426 <math>\pm</math> 119</b> | 2939 $\pm$ 28                    | <b>4.01 <math>\pm</math> 0.03</b> | <b>11783 <math>\pm</math> 129</b> |
|             | RefQD   | 1385 $\pm$ 23                    | 596 $\pm$ 10                     | 3.09 $\pm$ 0.13                   | 1840 $\pm$ 68                     |
|             | Ind-PPO | 3352 $\pm$ 102                   | <b>2945 <math>\pm</math> 41</b>  | 2.46 $\pm$ 0.11                   | 7252 $\pm$ 342                    |
| Seaquest    | RePPO   | <b>3382 <math>\pm</math> 465</b> | <b>1855 <math>\pm</math> 125</b> | <b>4.33 <math>\pm</math> 0.28</b> | <b>8100 <math>\pm</math> 830</b>  |
|             | RefQD   | 511 $\pm$ 19                     | 485 $\pm$ 5                      | 1.11 $\pm$ 0.05                   | 538 $\pm$ 26                      |
|             | Ind-PPO | 1907 $\pm$ 93                    | 1469 $\pm$ 90                    | 1.50 $\pm$ 0.10                   | 2173 $\pm$ 76                     |
| Ms. Pac-Man | RePPO   | 2588 $\pm$ 112                   | 2088 $\pm$ 36                    | 3.38 $\pm$ 0.14                   | <b>7061 <math>\pm</math> 369</b>  |
|             | RefQD   | 1176 $\pm$ 105                   | 474 $\pm$ 12                     | <b>3.65 <math>\pm</math> 0.03</b> | 1732 $\pm$ 42                     |
|             | Ind-PPO | <b>2730 <math>\pm</math> 132</b> | <b>2215 <math>\pm</math> 59</b>  | 2.30 $\pm$ 0.09                   | 5107 $\pm$ 255                    |

Figure 14 shows where each policy of the first seed plays, over eight games per policy. In Boxing, most Ind-PPO policies fight from the left side of the ring at one of two heights, whereas RePPO’s policies also fight from the right side of the ring and along its top and bottom edges. In Seaquest, Ind-PPO’s policies descend to the sea floor and stay there, while RePPO’s policies occupy different depths and horizontal positions, and its best policy remains just below the surface. In Ms. Pac-Man, all policies traverse the whole maze, so the figure instead shows when each dot is eaten. The policies of both methods start from the bottom of the maze, where the game begins, and differ in which regions they leave for last.

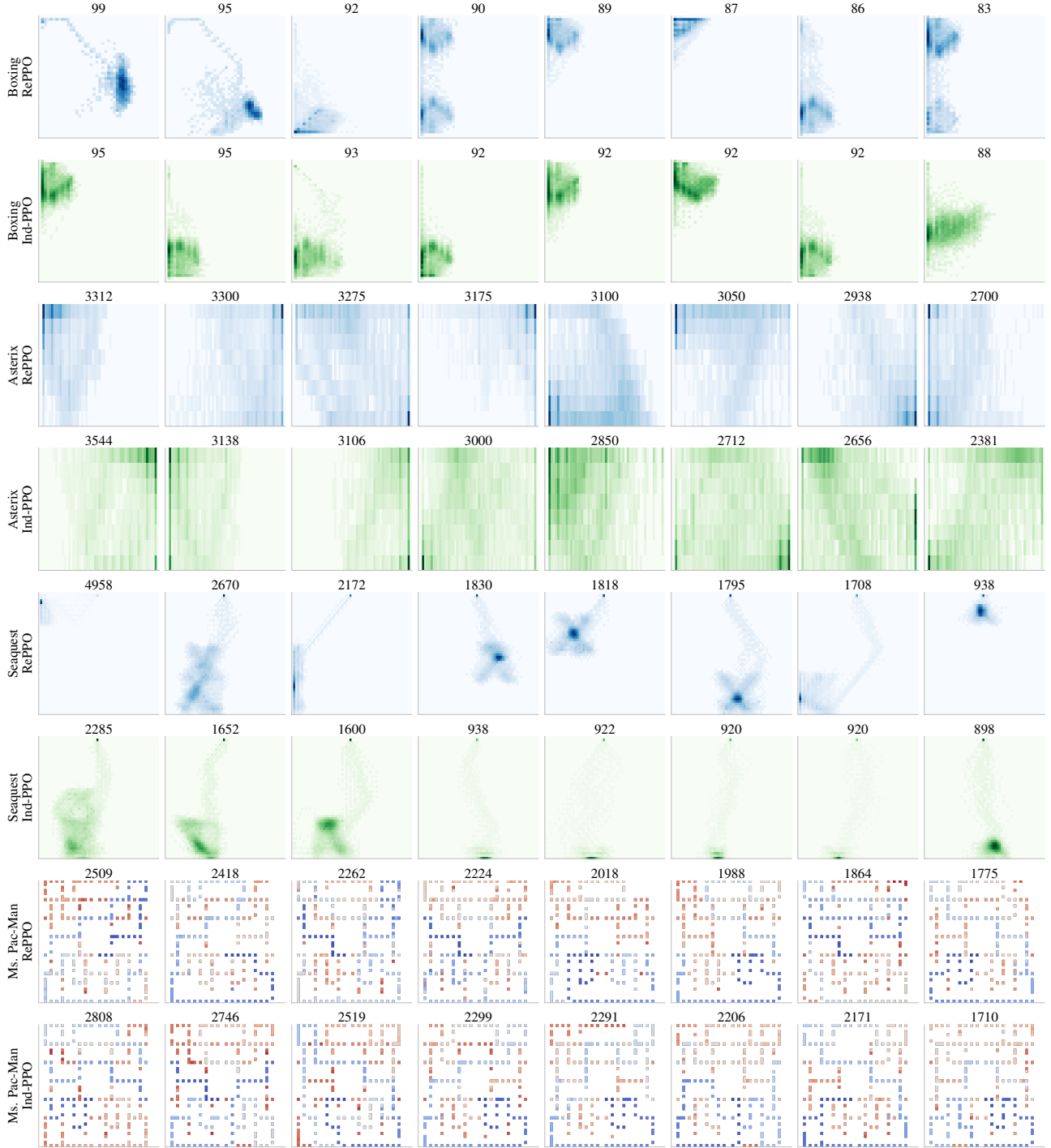


Figure 14. Behavior of the eight policies of the first seed of RePPO (blue) and Ind-PPO (green), sorted by score, which is shown above each panel. For Boxing, Asterix, and Seaquest, each panel is a heatmap of the player’s position on the screen. For Ms. Pac-Man, each panel colors the dots of the maze by when they are eaten, from early (blue) to late (red), as a fraction of the level.

## F. Fine-tuning large language models

### F.1. Task and data

We train on the 5584 problems of levels 3–5 of the MATH training set (Hendrycks et al., 2021) whose reference solution contains a boxed answer, as in SimpleRL-Zoo (Zeng et al., 2025). For checkpoint selection, we sample 384 problems of the same levels from the MATH test set, excluding MATH-500. The test set combines MATH-500 (Lightman et al., 2024) (500 problems), Minerva (Lewkowycz et al., 2022) (272), and the English text-only part of OlympiadBench (He et al., 2024) (675). The base models are Qwen3-1.7B-Base (Team, 2025) and Granite-4.1-3B-Base (IBM Granite Team, 2026). Every problem is accompanied by the instruction “Please reason step by step, and put your final answer within `\boxed{\}`.”, given as the system message of the chat template for Qwen3 and after the problem in a plain question–answer format for Granite.

**Fitness.** A solution scores  $r = 1$  if it ends within 4096 tokens and its last boxed expression matches the reference under Math-Verify, and  $r = 0$  otherwise. The training reward standardizes this score as  $(r - \mu_0)/\sigma_0$ , using its mean and standard deviation on the base model’s first batch.

**Descriptors.** Four raw features  $x_j$  are computed on the solution text up to its last boxed answer (Table 9). The descriptor corresponding to a solution is then  $d_j = r \tanh((x_j - \mu_j)/\sigma_j) \in [-1, 1]$ , where  $\mu_j$  is the mean of  $x_j$  over the base model’s correct solutions on its first batch and  $\sigma_j$  is a fixed scale shared by both models. Multiplying by  $r$  assigns the neutral value 0 to incorrect solutions, so that a member cannot move its descriptor by writing incorrect solutions in a particular style. Since a solution’s fitness and descriptors are revealed only when it is complete, both are terminal rewards in the token-level MDP.

Table 9. Raw features of the LLM descriptors and their scales  $\sigma_j$ .

| Descriptor   | Raw feature $x_j$                                                                      | $\sigma_j$ |
|--------------|----------------------------------------------------------------------------------------|------------|
| Code         | share of non-space characters inside code blocks                                       | 0.098      |
| Verification | self-verification episodes (“let me double-check”, “substituting back”) per 1000 words | 1.845      |
| Symbols      | share of non-code text inside $\LaTeX$ math delimiters                                 | 0.170      |
| Markdown     | share of non-empty prose lines that are list items, headers, or table rows             | 0.233      |

### F.2. Training

Each member is a LoRA adapter (Hu et al., 2022) of rank 32 on all attention and MLP projections of the frozen backbone, with its own critic: a rank-64 adapter on the same backbone with a linear head of  $1 + d = 5$  outputs. An actor adapter has 34.9 M parameters for Qwen3 and 62.3 M for Granite. Training follows the zero-RL recipe of Open-Reasoner-Zero (Hu et al., 2025): PPO with a learned critic,  $\gamma = \lambda = 1$ , and no KL penalty or entropy bonus. In every iteration, each member samples 8 solutions to each of the same 128 problems. Advantages are standardized per channel over each member’s full batch of 1024 solutions, and the diversity weights are computed with the shared implementation used in all other domains, from EMAs of each member’s batch-mean fitness and descriptors. Table 10 lists all hyperparameters. After every 10 iterations, we evaluate each population on the validation problems and select the checkpoint with the highest mean accuracy.

Table 10. Hyperparameters of LLM fine-tuning. Ind-PPO uses the same settings with the diversity weights set to zero.

| Parameter                               | Value                     | Parameter                    | Value                              |
|-----------------------------------------|---------------------------|------------------------------|------------------------------------|
| Population size $N$                     | 5                         | Actor / critic learning rate | $5 \times 10^{-5} / 10^{-4}$       |
| Iterations                              | 100                       | Critic-only warm-up          | 15 iterations                      |
| Problems $\times$ samples per iteration | $128 \times 8$ per member | Optimizer                    | AdamW (0.9, 0.95), no weight decay |
| Sampling (training)                     | temperature 1, top- $p$ 1 | Gradient clipping            | 1.0                                |
| Maximum response length                 | 4,096 tokens              | Neighbors $K$                | 4                                  |
| Discount $\gamma$ / GAE $\lambda$       | 1 / 1                     | Repulsion cap $C_{\max}$     | 0.5                                |
| PPO clip / value clip                   | 0.2 / 0.2                 | Kernel bandwidth $\ell^2$    | 0.5                                |
| Epochs / minibatch                      | 1 / 256 solutions         | EMA rates                    | 0.7                                |
| LoRA rank (actor / critic)              | 32 / 64                   | $\epsilon_f / \epsilon_b$    | 1 / 0.05                           |

### F.3. Baselines and evaluation

The prompting baselines prepend a style instruction to each problem. In the *corners* baseline, the five instructions combine extreme values of all four descriptors, chosen so that every pair of members differs in at least two of them. For example, “Solve the problem by writing Python code, verify your answer with an explicit check, explain mostly in words, and write in flowing paragraphs without headers or bullet lists.” In the *emphasis* baseline, four members each emphasize one descriptor, such as “Present the solution as a clearly structured markdown document, with headers and a bulleted or numbered list of steps.”, and the fifth combines markdown with verification.

At test time, every member samples 8 solutions per problem at temperature 0.6 and top- $p$  0.95, and the base model samples 40, matching a population’s total.  $\text{pass}@1$  averages correctness over members and samples. Population  $\text{pass}@5j$  draws  $j$  samples from each member and is computed with the unbiased estimator of Chen et al. (2021). Identifiability is the held-out accuracy of a logistic-regression classifier on TF-IDF features of word unigrams and bigrams, trained to predict which member wrote a solution on half of the problems and tested on the other half.

### F.4. Additional results

Table 11 reports accuracy by benchmark and  $\text{pass}@k$ . Fine-tuning mostly sharpens what Qwen3 can already solve, raising  $\text{pass}@40$  by 2 points, while Granite also learns to end its solutions more reliably, raising  $\text{pass}@40$  by 9 points. RePPO’s lower  $\text{pass}@1$  comes mainly from a single member in each population: the code-writing member of Qwen3, whose solutions are the longest and the outputs of the programs are guessed, and the Granite member that rarely uses an explicit reasoning block. Figure 15 shows that the instructions shift the style of the prompted members only moderately, even when they explicitly request extreme values of the descriptors.

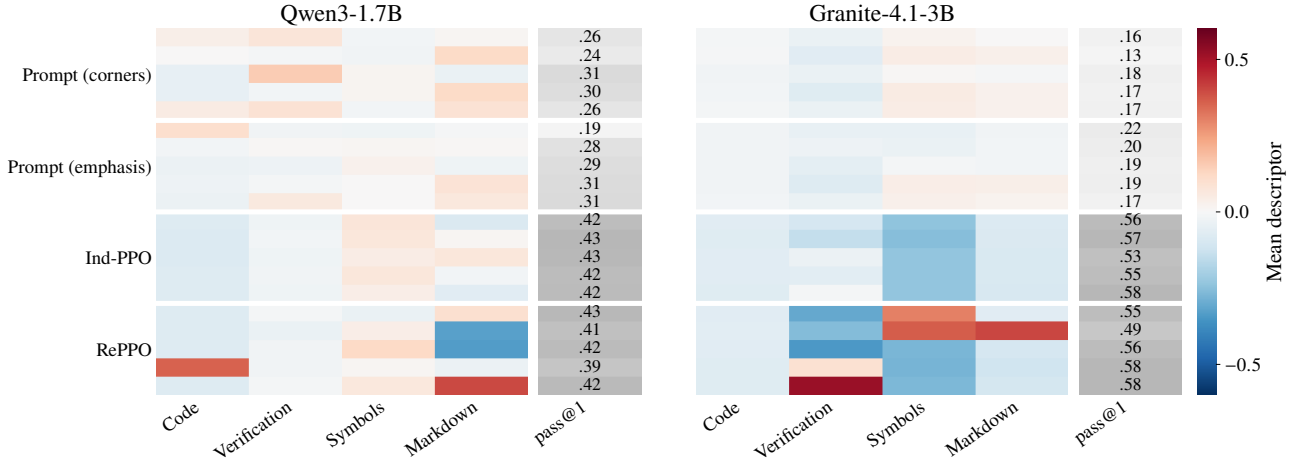


Figure 15. Mean descriptor and  $\text{pass}@1$  of every member, including the prompting baselines, as in Figure 3. Each block of five rows is one population.

Table 11. Accuracy by benchmark and population  $\text{pass}@k$  on the test problems.

| Model          | Method            | MATH-500 | Minerva | OlympiadBench | $\text{pass}@5$ | $\text{pass}@10$ | $\text{pass}@20$ | $\text{pass}@40$ |
|----------------|-------------------|----------|---------|---------------|-----------------|------------------|------------------|------------------|
| Qwen3-1.7B     | Base model        | 0.546    | 0.142   | 0.225         | 0.526           | 0.592            | 0.647            | 0.694            |
|                | Prompt (corners)  | 0.467    | 0.130   | 0.189         | 0.503           | 0.574            | 0.635            | 0.688            |
|                | Prompt (emphasis) | 0.470    | 0.121   | 0.192         | 0.506           | 0.578            | 0.639            | 0.694            |
|                | Ind-PPO           | 0.672    | 0.274   | 0.299         | 0.589           | 0.640            | 0.681            | 0.713            |
|                | RePPO             | 0.664    | 0.266   | 0.287         | 0.588           | 0.641            | 0.683            | 0.715            |
| Granite-4.1-3B | Base model        | 0.392    | 0.072   | 0.181         | 0.500           | 0.588            | 0.654            | 0.704            |
|                | Prompt (corners)  | 0.282    | 0.057   | 0.118         | 0.424           | 0.530            | 0.616            | 0.681            |
|                | Prompt (emphasis) | 0.332    | 0.064   | 0.143         | 0.469           | 0.571            | 0.647            | 0.700            |
|                | Ind-PPO           | 0.776    | 0.296   | 0.506         | 0.712           | 0.744            | 0.771            | 0.793            |
|                | RePPO             | 0.793    | 0.289   | 0.475         | 0.707           | 0.740            | 0.767            | 0.791            |

### F.5. Qualitative examples

Reading the solutions confirms that each RePPO member commits to a consistent style across problems. On Qwen3, the code-writing member includes a Python block in 92% of its solutions, almost always after a complete solution in prose, and another member formats 99% of its solutions as numbered lists. Two members write in plain prose and resemble each other closely, which matches their neighboring descriptors in Figure 3. On Granite, every trained member inherits the base model’s habit of reasoning in a block delimited by `<think>` before writing a final solution, and the members differ mainly within that block: one re-checks intermediate results as a reflex, one typesets its reasoning in  $\text{\LaTeX}$ , and one skips the block on most problems in favor of a structured write-up. Ind-PPO’s members are hard to tell apart in both models.

The styles are not always desirable. The code blocks of Qwen3’s code-writing member are followed by output blocks that the model writes itself, since no interpreter is available, and the stated outputs agree with what the code would actually print in only half of the cases. Some of the Granite member’s verification episodes repeat the same check several times. Both behaviors maximize the descriptors as defined, which shows that RePPO diversifies whatever the descriptors measure, so that their design determines which kind of diversity is obtained.

The boxes below show complete, unedited solutions to two MATH-500 problems for each model: the first of the eight test samples of every RePPO member and of two Ind-PPO members. The header of each box gives the four descriptors of that solution. On the seating problem, every member follows the same strategy, so the differences between the solutions are purely in presentation. Granite’s solutions follow a template inherited from the base model: an early boxed guess, often a placeholder, then reasoning inside a block delimited by `<think>`, shown in gray, and finally a written solution.

#### Qwen3-1.7B: Seating with an adjacency constraint

In how many ways can 6 girls and 2 boys sit in a row if the 2 boys insist on sitting next to each other?

Counting & Probability, level 3; answer 10,080.

#### RePPO · Member 0

correct, 399 tokens

code  $-0.19$  verification  $-0.18$  symbols  $-0.91$  markdown  $+0.89$

To solve this problem, we need to consider the arrangement of the girls and the boys such that the two boys sit next to each other. Here’s a step-by-step reasoning:

1. **Treat the two boys as a single unit:** Since the two boys must sit next to each other, we can think of them as a single “block” or “super person”. This reduces the problem to arranging 7 units (6 girls + 1 block of 2 boys).
2. **Arrange the 7 units:** The number of ways to arrange 7 units in a row is given by the factorial of 7, which is  $7!$ . Calculating this, we get:

$$7! = 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 5040$$

3. **Arrange the boys within the block:** Within the block, the two boys can switch places. There are  $2!$  (2 factorial) ways to arrange the two boys within their block. Calculating this, we get:

$$2! = 2 \times 1 = 2$$

4. **Combine the arrangements:** The total number of ways to arrange the 6 girls and 2 boys such that the two boys sit next to each other is the product of the number of ways to arrange the 7 units and the number of ways to arrange the boys within their block. Therefore, we multiply the results from steps 2 and 3:

$$7! \times 2! = 5040 \times 2 = 10080$$

Thus, the number of ways to arrange 6 girls and 2 boys in a row such that the two boys sit next to each other is 10080.

**RePPO · Member 1**

correct, 290 tokens

code −0.19   verification −0.18   symbols −0.74   markdown −0.89

To solve the problem, we can treat the two boys as a single “block” or “unit”. This means we have 7 units to arrange: the block of 2 boys and the 6 girls. The number of ways to arrange these 7 units is  $7!$  (7 factorial).

After arranging the 7 units, the 2 boys within their block can switch places. There are  $2!$  (2 factorial) ways to arrange the boys within their block.

Therefore, the total number of ways to arrange the 6 girls and 2 boys such that the 2 boys sit next to each other is:

$$7! \times 2!$$

Calculating  $7!$ :

$$7! = 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 5040$$

Calculating  $2!$ :

$$2! = 2 \times 1 = 2$$

Multiplying these together gives:

$$7! \times 2! = 5040 \times 2 = 10080$$

Thus, the number of ways the 6 girls and 2 boys can sit in a row if the 2 boys insist on sitting next to each other is:

$$\boxed{10080}$$

**RePPO · Member 2**

correct, 254 tokens

code −0.19   verification −0.18   symbols −0.64   markdown −0.89

To solve this problem, we can treat the two boys as a single “block” or “unit”. This means we have 7 units to arrange: the block of 2 boys and the 6 girls. The number of ways to arrange these 7 units is  $7!$  (7 factorial).

After arranging the 7 units, the 2 boys within their block can be arranged in  $2!$  (2 factorial) ways. Therefore, the total number of arrangements is:

$$7! \times 2!$$

First, calculate  $7!$ :

$$7! = 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 5040$$

Next, calculate  $2!$ :

$$2! = 2 \times 1 = 2$$

Now, multiply these two results together:

$$7! \times 2! = 5040 \times 2 = 10080$$

Thus, the number of ways the 6 girls and 2 boys can sit in a row if the 2 boys insist on sitting next to each other is  $\boxed{10080}$ .

**RePPO · Member 3**

correct, 388 tokens

code +0.99   verification −0.18   symbols −0.86   markdown +0.19

To solve this problem, we need to consider the boys as a single “block” since they insist on sitting next to each other. This means we can treat the two boys as one unit, and then we need to arrange this block along with the 6 girls.

Here’s the step-by-step reasoning:

1. Treat the two boys as a single unit. This means we now have 7 units to arrange: the block of two boys and the 6 girls.
2. The number of ways to arrange these 7 units is  $7!$  (7 factorial), which is  $7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 5040$ .
3. Within the block, the two boys can switch places, so there are  $2!$  (2 factorial) ways to arrange the two boys within their block, which is  $2 \times 1 = 2$ .

So the total number of arrangements is  $7! \times 2! = 5040 \times 2 = 10080$ .

Let’s confirm this with Python code. `python`

```
import math

# Number of ways to arrange 7 units (1 block of 2 boys and 6 girls)
arrangements_of_units = math.factorial(7)

# Number of ways to arrange the 2 boys within their block
arrangements_within_block = math.factorial(2)

# Total number of arrangements
total_arrangements = arrangements_of_units * arrangements_within_block
print(total_arrangements)
```

output

```
| 10080
```

The total number of ways to arrange 6 girls and 2 boys in a row, with the 2 boys insisting on sitting next to each other, is 10080.

#### RePPO · Member 4

correct, 330 tokens

code −0.19 verification −0.18 symbols −0.85 markdown +0.89

Let's break down the problem step by step:

1. **Treat the 2 boys as a single unit:** Since the 2 boys must sit next to each other, we can consider them as one “block” or “unit.” This means we now have 7 units to arrange: the 6 girls and the 1 block of 2 boys.
2. **Arrange the 7 units:** The number of ways to arrange these 7 units in a row is  $7!$  (7 factorial).

$$7! = 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 5040$$

3. **Arrange the 2 boys within their block:** Within the block, the 2 boys can switch places. There are  $2!$  (2 factorial) ways to arrange the 2 boys within their block.

$$2! = 2 \times 1 = 2$$

4. **Calculate the total number of arrangements:** Multiply the number of ways to arrange the 7 units by the number of ways to arrange the 2 boys within their block.

$$7! \times 2! = 5040 \times 2 = 10080$$

Therefore, the total number of ways the 6 girls and 2 boys can sit in a row with the 2 boys sitting next to each other is:

10080

#### Ind-PPO · Member 0

correct, 359 tokens

code −0.19 verification −0.18 symbols −0.86 markdown +0.91

To solve the problem of determining how many ways 6 girls and 2 boys can sit in a row with the condition that the 2 boys insist on sitting next to each other, we can treat the two boys as a single “block” or “unit”. This means we are effectively arranging 7 units (the block of 2 boys and the 6 girls).

1. **Treat the boys as a single unit:**
  - We have 7 units to arrange: the block of 2 boys and the 6 girls.
  - The number of ways to arrange these 7 units is  $7!$  (7 factorial).
2. **Arrange the boys within their block:**
  - The two boys can switch places within their block. So, there are  $2!$  (2 factorial) ways to arrange the boys within their block.
3. **Calculate the total number of arrangements:**

- The total number of ways to arrange the 7 units and the boys within their block is  $7! \times 2!$ .

Let's calculate this step by step:

- $7! = 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 5040$
- $2! = 2 \times 1 = 2$

So, the total number of arrangements is:

$$7! \times 2! = 5040 \times 2 = 10080$$

Therefore, the number of ways in which 6 girls and 2 boys can sit in a row if the 2 boys insist on sitting next to each other is 10080.

#### Ind-PPO · Member 1

correct, 407 tokens

code -0.19 verification -0.18 symbols -0.92 markdown +0.98

To solve this problem, we can treat the two boys as a single entity since they insist on sitting next to each other. This way, we can simplify the problem by considering the combined unit of the two boys as one "block."

**Step 1: Treat the two boys as a single block** - If the two boys must sit next to each other, we can consider them as one "block." This block can be arranged in 2 ways: Boy A first and Boy B second, or Boy B first and Boy A second.

**Step 2: Count the total number of entities to arrange** - Along with the two boys block, we have the 6 girls. So, we now have: - 1 two-boy block - 6 girls - This gives a total of  $1 + 6 = 7$  entities to arrange.

**Step 3: Arrange the entities** - The number of ways to arrange 7 entities in a row is  $7!$  (7 factorial). -  $7! = 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 5040$ .

**Step 4: Arrange the two boys within the block** - As mentioned earlier, the two boys can switch places within their block. So, there are 2 ways to arrange the two boys within their block.

**Step 5: Calculate the total number of arrangements** - Multiply the number of ways to arrange the entities by the number of ways to arrange the boys within their block:

$$7! \times 2 = 5040 \times 2 = 10080$$

**Final Answer** The total number of ways the 6 girls and 2 boys can sit in a row, with the two boys sitting next to each other, is:

10080

#### Qwen3-1.7B: The area of an isosceles triangle

What is the number of square units in the area of a triangle whose sides measure 5, 5 and 6 units?

Geometry, level 2; answer 12.

#### RePPO · Member 0

correct, 437 tokens

code -0.19 verification -0.18 symbols -0.09 markdown -0.13

To find the area of a triangle with sides measuring 5, 5, and 6 units, we can use Heron's formula. Heron's formula allows us to calculate the area of a triangle when we know the lengths of all three sides. The formula is:

$$\text{Area} = \sqrt{s(s-a)(s-b)(s-c)}$$

where  $s$  is the semi-perimeter of the triangle, and  $a$ ,  $b$ , and  $c$  are the lengths of the sides.

**Step 1: Calculate the semi-perimeter**  $s$  The semi-perimeter  $s$  is half of the perimeter of the triangle. The perimeter is the sum of the lengths of the sides. So, we have:

$$s = \frac{a + b + c}{2}$$

Given the sides  $a = 5$ ,  $b = 5$ , and  $c = 6$ , we can substitute these values into the formula:

$$s = \frac{5 + 5 + 6}{2} = \frac{16}{2} = 8$$

**Step 2: Apply Heron's formula** Now that we have the semi-perimeter  $s = 8$ , we can substitute  $a = 5$ ,  $b = 5$ ,  $c = 6$ , and  $s = 8$  into Heron's formula:

$$\text{Area} = \sqrt{s(s-a)(s-b)(s-c)} = \sqrt{8(8-5)(8-5)(8-6)}$$

Simplify the terms inside the square root:

$$\text{Area} = \sqrt{8 \times 3 \times 3 \times 2}$$

$$\text{Area} = \sqrt{144}$$

$$\text{Area} = 12$$

**Final Answer** The area of the triangle is 12 square units.

#### RePPO · Member 1

correct, 297 tokens

code -0.19 verification -0.18 symbols +0.24 markdown -0.89

To find the area of a triangle with sides measuring 5, 5, and 6 units, we can use Heron's formula. Heron's formula states that the area  $A$  of a triangle with sides of lengths  $a$ ,  $b$ , and  $c$  is given by:

$$A = \sqrt{s(s-a)(s-b)(s-c)}$$

where  $s$  is the semi-perimeter of the triangle, calculated as:

$$s = \frac{a + b + c}{2}$$

For our triangle, the sides are  $a = 5$ ,  $b = 5$ , and  $c = 6$ . First, we calculate the semi-perimeter  $s$ :

$$s = \frac{5 + 5 + 6}{2} = \frac{16}{2} = 8$$

Next, we substitute  $s$ ,  $a$ ,  $b$ , and  $c$  into Heron's formula:

$$A = \sqrt{8(8-5)(8-5)(8-6)}$$

$$A = \sqrt{8 \cdot 3 \cdot 3 \cdot 2}$$

$$A = \sqrt{144}$$

$$A = 12$$

Therefore, the area of the triangle is 12 square units.

#### RePPO · Member 2

correct, 313 tokens

code -0.19 verification -0.18 symbols +0.40 markdown -0.89

To find the area of a triangle with sides measuring 5, 5, and 6 units, we can use Heron's formula. Heron's formula states that the area  $A$  of a triangle with sides  $a$ ,  $b$ , and  $c$  is given by:

$$A = \sqrt{s(s-a)(s-b)(s-c)}$$

where  $s$  is the semi-perimeter of the triangle, calculated as:

$$s = \frac{a + b + c}{2}$$

For our triangle, the sides are  $a = 5$ ,  $b = 5$ , and  $c = 6$ . First, we calculate the semi-perimeter  $s$ :

$$s = \frac{5 + 5 + 6}{2} = \frac{16}{2} = 8$$

Next, we substitute  $s$ ,  $a$ ,  $b$ , and  $c$  into Heron's formula:

$$A = \sqrt{8(8 - 5)(8 - 5)(8 - 6)}$$

$$A = \sqrt{8 \cdot 3 \cdot 3 \cdot 2}$$

$$A = \sqrt{8 \cdot 9 \cdot 2}$$

$$A = \sqrt{144}$$

$$A = 12$$

Therefore, the area of the triangle is 12 square units.

### RePPO · Member 3

correct, 388 tokens

code +0.95 verification -0.18 symbols +0.21 markdown -0.89

To find the area of a triangle with sides measuring 5, 5, and 6 units, we can use Heron's formula. Heron's formula states that the area of a triangle with sides  $a$ ,  $b$ , and  $c$  is given by:

$$\text{Area} = \sqrt{s(s - a)(s - b)(s - c)}$$

where  $s$  is the semi-perimeter of the triangle:

$$s = \frac{a + b + c}{2}$$

For our triangle,  $a = 5$ ,  $b = 5$ , and  $c = 6$ . So, the semi-perimeter  $s$  is:

$$s = \frac{5 + 5 + 6}{2} = \frac{16}{2} = 8$$

Now we can substitute  $a$ ,  $b$ ,  $c$ , and  $s$  into Heron's formula:

$$\text{Area} = \sqrt{8(8 - 5)(8 - 5)(8 - 6)} = \sqrt{8 \cdot 3 \cdot 3 \cdot 2} = \sqrt{144} = 12$$

Let's confirm this with Python code. `python`

```
import math

# Side lengths
a = 5
b = 5
c = 6

# Semi-perimeter
s = (a + b + c) / 2

# Area using Heron's formula
area = math.sqrt(s * (s - a) * (s - b) * (s - c))
print(area)
```

output

```
| 12.0
```

The area of the triangle with sides measuring 5, 5, and 6 units is 12.

**RePPO · Member 4**

correct, 529 tokens

code -0.19 verification +0.94 symbols -0.58 markdown +0.98

Sure, let's break this down step by step:

- 1. Identify the type of triangle:** The sides given are 5, 5, and 6 units. This is an isosceles triangle because two sides are equal (5 units each).
- 2. Determine if it's a right triangle:** To check if it's a right triangle, we can use the Pythagorean theorem. For a right triangle with legs  $a$  and  $b$  and hypotenuse  $c$ , the theorem states  $a^2 + b^2 = c^2$ .
  - Here,  $a = 5$ ,  $b = 5$ , and  $c = 6$ .
  - Calculate  $a^2 + b^2 = 5^2 + 5^2 = 25 + 25 = 50$ .
  - Calculate  $c^2 = 6^2 = 36$ .
- 3. Check the equality:**  $50 \neq 36$ . Therefore, it is not a right triangle.
- 4. Calculate the area:** Since it is an isosceles triangle but not a right triangle, we use the formula for the area of a triangle:  $\text{Area} = \frac{1}{2} \times \text{base} \times \text{height}$ .
  - We need to find the height. Let's drop a perpendicular from the vertex opposite the base (6 units) to the base, splitting the base into two equal segments of 3 units each.
  - This forms two right triangles with legs of 3 units and the height  $h$ , and the hypotenuse of 5 units.
  - Using the Pythagorean theorem again:  $3^2 + h^2 = 5^2$ .
  - Calculate  $9 + h^2 = 25$ .
  - Solve for  $h^2$ :  $h^2 = 25 - 9 = 16$ .
  - Therefore,  $h = \sqrt{16} = 4$ .
- 5. Calculate the area:** Now, use the height to find the area.
  - $\text{Area} = \frac{1}{2} \times 6 \times 4 = \frac{1}{2} \times 24 = 12$ .

So, the number of square units in the area of the triangle is 12.

**Ind-PPO · Member 0**

correct, 412 tokens

code -0.19 verification -0.18 symbols -0.11 markdown -0.33

To find the area of a triangle with sides measuring 5, 5, and 6 units, we can use **Heron's Formula**. Here's the step-by-step reasoning:

**Step 1: Identify the type of triangle** The triangle has two sides of equal length (5 units each) and a third side of 6 units. This is an **isosceles triangle**.

**Step 2: Calculate the semi-perimeter ( $s$ )** The semi-perimeter is half the perimeter of the triangle. The perimeter is the sum of all sides.

$$\text{Perimeter} = 5 + 5 + 6 = 16 \text{ units}$$

$$s = \frac{\text{Perimeter}}{2} = \frac{16}{2} = 8 \text{ units}$$

**Step 3: Apply Heron's Formula** Heron's Formula states that the area ( $A$ ) of a triangle is:

$$A = \sqrt{s(s-a)(s-b)(s-c)}$$

where  $a$ ,  $b$ , and  $c$  are the lengths of the sides of the triangle.

For this triangle:

$$a = 5, \quad b = 5, \quad c = 6$$

Substitute the values into Heron's Formula:

$$A = \sqrt{8(8-5)(8-5)(8-6)}$$

$$A = \sqrt{8 \times 3 \times 3 \times 2}$$

$$A = \sqrt{8 \times 9 \times 2}$$

$$A = \sqrt{144}$$

$$A = 12 \text{ square units}$$

**Final Answer:** The area of the triangle is 12 square units.

#### Ind-PPO · Member 1

correct, 386 tokens

code -0.19 verification -0.18 symbols -0.26 markdown -0.27

To determine the area of a triangle with sides measuring 5, 5, and 6 units, we can use **Heron's Formula**. Here's a step-by-step breakdown:

**Step 1: Identify the type of triangle** The triangle has two sides of equal length (5 units each) and a third side of 6 units. This is an **isosceles triangle**.

**Step 2: Calculate the semi-perimeter (s)** The semi-perimeter  $s$  is half the perimeter of the triangle. The perimeter  $P$  is the sum of all sides:

$$P = 5 + 5 + 6 = 16 \text{ units}$$

$$s = \frac{P}{2} = \frac{16}{2} = 8 \text{ units}$$

**Step 3: Apply Heron's Formula** Heron's Formula states that the area  $A$  of a triangle is:

$$A = \sqrt{s(s-a)(s-b)(s-c)}$$

where  $a$ ,  $b$ , and  $c$  are the lengths of the sides of the triangle.

Substitute the known values:

$$a = 5, \quad b = 5, \quad c = 6, \quad s = 8$$

$$A = \sqrt{8(8-5)(8-5)(8-6)}$$

$$A = \sqrt{8 \cdot 3 \cdot 3 \cdot 2}$$

$$A = \sqrt{144}$$

$$A = 12 \text{ square units}$$

**Final Answer:** The area of the triangle is 12.

**Granite-4.1-3B: Seating with an adjacency constraint**

In how many ways can 6 girls and 2 boys sit in a row if the 2 boys insist on sitting next to each other?

Counting & Probability, level 3; answer 10,080.

**RePPO · Member 0**

correct, 329 tokens

code −0.13 verification −0.66 symbols −0.45 markdown −0.24

`answer`.

Where [answer] is just the final number or expression that solves the problem. If the question is a multiple choice question, [answer] should be the letter indicating your correct response (e.g. `\text{A}` or `\text{B}`).

`<think>`We are asked: “In how many ways can 6 girls and 2 boys sit in a row if the 2 boys insist on sitting next to each other?” We’ll solve: Consider the two boys as a block. Then we have 6 girls + 1 block = 7 “objects” to arrange. The number of ways to arrange these 7 objects is  $7! = 5040$ . However, the two boys inside the block can be arranged in  $2! = 2$  ways. So total arrangements =  $7! * 2! = 5040 * 2 = 10080$ .

Thus final answer: `\boxed{10080}`.

I’ll produce answer: `\boxed{10080}`.

I’ll produce answer: `\boxed{10080}`. `</think>`

Step 1: Treat the 2 boys as a single unit (block). This gives us 6 girls + 1 block = 7 units to arrange.

Step 2: The 7 units can be arranged in  $7!$  ways.

Step 3: Within the block, the 2 boys can be arranged in  $2!$  ways.

Step 4: Multiply the two results:  $7! \times 2! = 5040 \times 2 = 10080$ .

`\boxed{10080}`

**RePPO · Member 1**

correct, 303 tokens

code −0.13 verification −0.66 symbols +0.31 markdown +0.99

To solve the problem, follow these steps:

1. **Treat the 2 Boys as a Single Unit:** Since the 2 boys must sit next to each other, we can consider them as a single “block” or unit. This reduces the problem to arranging 7 units: the “boy block” and the 6 girls.
2. **Calculate the Arrangements of the 7 Units:** The 7 units can be arranged in  $7!$  ways.

$$7! = 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 5040$$

3. **Arrange the 2 Boys within the “Boy Block”:** Within the “boy block”, the 2 boys can be arranged among themselves in  $2!$  ways.

$$2! = 2 \times 1 = 2$$

4. **Calculate the Total Number of Arrangements:** Multiply the number of ways to arrange the 7 units by the number of ways to arrange the 2 boys within their block.

$$7! \times 2! = 5040 \times 2 = 10080$$

Thus, the total number of ways the 6 girls and 2 boys can sit in a row with the 2 boys sitting next to each other is `\boxed{10080}`.

**RePPO · Member 2**

correct, 490 tokens

code −0.13 verification −0.66 symbols −0.54 markdown −0.24

`\boxed{1440}`

`<think>`We are asked: “In how many ways can 6 girls and 2 boys sit in a row if the 2 boys insist on sitting next to each

$$\boxed{10080}$$
 $\boxed{10080}$

**RePPO · Member 4**

correct, 702 tokens

code -0.13 verification +1.00 symbols -0.54 markdown -0.24

 $\boxed{720}$ 

*<think>*We are asked: “In how many ways can 6 girls and 2 boys sit in a row if the 2 boys insist on sitting next to each other?” We need to count arrangements. Let’s check: We have 6 girls and 2 boys. The condition is that the two boys must sit next to each other. We can treat the two boys as a single block. Then we have 7 objects: 6 girls + 1 block of 2 boys. They can be arranged in  $7!$  ways. But within the block, the two boys can be arranged in  $2!$  ways. So total =  $7! * 2!$ . Let’s compute:  $7! = 5040$ , times  $2 = 10080$ . Wait check: Actually,  $7! = 5040$ , times  $2 = 10080$ . But let me check:  $7! = 5040$ , multiplied by 2 equals 10080. So answer is 10080. But wait, check: Actually, the number is  $7! * 2 = 50402 = 10080$ . But wait, let me check: Actually, I need to check: 6 girls and 2 boys. The two boys must be adjacent. So yes, treat them as one block. So answer =  $7! * 2 = 50402 = 10080$ . But let me check: Alternatively, we can consider 6 girls arranged in  $6!$  ways, and then insert the block of 2 boys in 7 positions, times  $2!$  for ordering the boys. That gives  $6! * 7 * 2 = 720 * 7 * 2 = 720 * 14 = 10080$ . So answer is 10080.

Wait, check:  $6! = 720$ , times  $7 = 5040$ , times  $2 = 10080$ . So answer is 10080. So answer is  $\boxed{10080}$ . But wait, let me check: The problem said: “In how many ways can 6 girls and 2 boys sit in a row if the 2 boys insist on sitting next to each other?” So answer is  $7! * 2 = 10080$ . So final answer is  $\boxed{10080}$ .

But wait, let me check: Actually, the answer is 10080. So final answer:  $\boxed{10080}$ .

I’ll produce answer with final answer in the required format:  $\boxed{10080}$ . But instructions: “Make sure to put the answer (and only answer) inside  $\boxed{\quad}$ ”. So final answer:  $\boxed{10080}$ . But instructions: “Please reason step by step, and put your final answer within  $\boxed{\quad}$ ”. I’ll produce answer with step by step reasoning. *</think>*

Step 1: Since the 2 boys must sit together, treat them as a single unit. This gives us 6 girls + 1 block = 7 units.

Step 2: The 7 units can be arranged in  $7!$  ways.

Step 3: Within the block, the 2 boys can be arranged in  $2!$  ways.

Step 4: Multiply the two results:  $7! \times 2! = 5040 \times 2 = 10080$ .

 $\boxed{10080}$ 
**Ind-PPO · Member 0**

correct, 323 tokens

code -0.13 verification -0.66 symbols -0.54 markdown -0.24

 $\boxed{1440}$ 

*<think>*We are asked: “In how many ways can 6 girls and 2 boys sit in a row if the 2 boys insist on sitting next to each other?” We want answer inside  $\boxed{\quad}$ .

Let’s solve: We have 6 girls and 2 boys, total 8 persons. The condition is that the two boys must sit next to each other. We can treat the pair of boys as a single unit. Then we have: unit for boys, plus 6 girls, total 7 units. They can be arranged in  $7!$  ways. And the two boys inside the unit can be arranged in  $2!$  ways. So answer is  $7! * 2! = 5040 * 2 = 10080$ . So answer:  $\boxed{10080}$ . But the answer should be inside  $\boxed{\quad}$ .

I’ll produce answer:  $\boxed{10080}$ . *</think>*

Step 1: Treat the 2 boys as a single unit since they must sit together. This gives us 6 girls + 1 “boy unit” = 7 units in total.

Step 2: Arrange these 7 units in a row. There are  $7!$  ways to do this.

Step 3: Within the “boy unit,” the 2 boys can be arranged in  $2!$  ways.

Step 4: Multiply the number of arrangements from Step 2 and Step 3:  $7! \times 2! = 5040 \times 2 = 10080$ .

 $\boxed{10080}$ 
**Ind-PPO · Member 1**

correct, 375 tokens

code -0.13 verification -0.66 symbols -0.54 markdown -0.24

 $\boxed{1440}$ 

*<think>*We are asked: “In how many ways can 6 girls and 2 boys sit in a row if the 2 boys insist on sitting next to each other?” We need to provide answer inside  $\boxed{\quad}$ .

Let’s solve: We have 6 girls and 2 boys. The two boys must sit next to each other. We can treat the two boys as a block. Then we have 7 objects: the block (2 boys together) plus 6 girls. The number of ways to arrange these 7 objects is  $7! = 5040$ . But within the block, the two boys can be arranged in  $2! = 2$  ways. So total number of arrangements is  $7! * 2! = 5040 * 2 = 10080$ . So answer:  $\boxed{10080}$ .

Wait check:  $7! = 5040$ , multiplied by 2 equals 10080. So answer:  $\boxed{10080}$ .

I'll produce answer with step-by-step reasoning. We'll produce answer:  $\boxed{10080}$ . We'll produce answer with final answer inside  $\boxed{\phantom{00000}}$ .

I'll produce answer:  $\boxed{10080}$ . We'll produce answer with final answer inside  $\boxed{\phantom{00000}}$ .  $\text{</think>}$

Step 1: Treat the two boys as a single unit since they must sit together. This gives us 6 girls + 1 block = 7 units to arrange.

Step 2: The 7 units can be arranged in  $7!$  ways.

Step 3: Within the block, the 2 boys can be arranged in  $2!$  ways.

Step 4: Multiply the two results:  $7! \times 2! = 5040 \times 2 = 10080$ .

$\boxed{10080}$

### Granite-4.1-3B: The area of an isosceles triangle

What is the number of square units in the area of a triangle whose sides measure 5, 5 and 6 units?

Geometry, level 2; answer 12.

### RePPO · Member 0

code  $-0.13$  verification  $-0.66$  symbols  $+0.99$  markdown  $-0.24$

correct, 217 tokens

We can use Heron's formula to find the area of the triangle. First, we calculate the semi-perimeter  $s$ :

$$s = \frac{5 + 5 + 6}{2} = \frac{16}{2} = 8.$$

Now, we apply Heron's formula:

$$\text{Area} = \sqrt{s(s-a)(s-b)(s-c)},$$

where  $a = 5$ ,  $b = 5$ , and  $c = 6$ . Plugging in the values:

$$\text{Area} = \sqrt{8(8-5)(8-5)(8-6)} = \sqrt{8 \times 3 \times 3 \times 2}.$$

$$\text{Area} = \sqrt{8 \times 9 \times 2} = \sqrt{144} = 12.$$

Thus, the area of the triangle is  $\boxed{12}$  square units.

### RePPO · Member 1

code  $-0.13$  verification  $-0.66$  symbols  $+0.96$  markdown  $+0.97$

correct, 346 tokens

To find the area of a triangle with sides  $a$ ,  $b$ , and  $c$ , we can use Heron's formula. Heron's formula states that the area  $A$  of a triangle with sides  $a$ ,  $b$ , and  $c$  is given by:

$$A = \sqrt{s(s-a)(s-b)(s-c)}$$

where  $s$  is the semi-perimeter of the triangle, calculated as:

$$s = \frac{a + b + c}{2}$$

For the given triangle with sides  $a = 5$ ,  $b = 5$ , and  $c = 6$ :

1. Calculate the semi-perimeter  $s$ :

$$s = \frac{5 + 5 + 6}{2} = \frac{16}{2} = 8$$

2. Substitute  $s$ ,  $a$ ,  $b$ , and  $c$  into Heron's formula:

$$A = \sqrt{8(8-5)(8-5)(8-6)}$$

3. Simplify inside the square root:

$$A = \sqrt{8 \times 3 \times 3 \times 2}$$

4. Calculate the product:

$$A = \sqrt{8 \times 3 \times 3 \times 2} = \sqrt{144}$$

5. Find the square root:

$$A = 12$$

Thus, the area of the triangle is 12 square units.

#### RePPO · Member 2

correct, 401 tokens

code −0.13 verification −0.66 symbols −0.54 markdown −0.24

`\boxed{12}`

`<think>`We are asked: “Solve the following math problem. Make sure to put the answer (and only answer) inside `\boxed{}`.”

What is the number of square units in the area of a triangle whose sides measure 5, 5 and 6 units?”

We need step-by-step reasoning, final answer inside `\boxed{}`.

We have a triangle with sides 5, 5, 6. This is an isosceles triangle with two equal sides 5 and base 6. We can use Heron’s formula. Let  $a = 5$ ,  $b = 5$ ,  $c = 6$ . Then semiperimeter  $s = (5+5+6)/2 = 16/2 = 8$ . Then  $\text{area} = \sqrt{s(s-a)(s-b)(s-c)} = \sqrt{8(8-5)(8-5)(8-6)} = \sqrt{8(3)(3)(2)} = \sqrt{144} = 12$ . So answer: `\boxed{12}`. `</think>`

Let the sides of the triangle be  $a = 5$ ,  $b = 5$ , and  $c = 6$ . First, compute the semiperimeter:

$$s = (5 + 5 + 6) / 2 = 16 / 2 = 8$$

Now, apply Heron’s formula for the area:

$$\text{Area} = \sqrt{s(s-a)(s-b)(s-c)} = \sqrt{8(8-5)(8-5)(8-6)} = \sqrt{8 \times 3 \times 3 \times 2} = \sqrt{144} = 12$$

Thus, the area of the triangle is `\boxed{12}`.

#### RePPO · Member 3

correct, 358 tokens

code −0.13 verification −0.66 symbols −0.54 markdown −0.24

`\boxed{12}`

`<think>`We are given triangle with sides 5, 5, 6. We want area. Let’s solve: Use Heron’s formula. Let sides:  $a=5$ ,  $b=5$ ,  $c=6$ . semiperimeter  $s = (5+5+6)/2 = 16/2 = 8$ . Then  $\text{area} = \sqrt{s(s-a)(s-b)(s-c)} = \sqrt{8(8-5)(8-5)(8-6)} = \sqrt{8(3)(3)(2)} = \sqrt{144} = 12$ . So answer is `\boxed{12}`. But careful: The problem says “put the answer (and only answer) inside `\boxed{}`.” But the instructions say “put your final answer within `\boxed{}`.” So answer: `\boxed{12}`. But check: Actually the answer is 12.

I’ll produce step-by-step explanation.

I’ll produce final answer: `\boxed{12}`. `</think>`

We are given a triangle with sides 5, 5, and 6. First, we compute the semiperimeter (s):

$$s = (5 + 5 + 6) / 2 = 16 / 2 = 8$$

Next, we use Heron’s formula for the area (A):

$$A = \sqrt{s(s-a)(s-b)(s-c)} = \sqrt{8 \times 3 \times 3 \times 2} = \sqrt{144} = 12$$

Thus, the area of the triangle is `\boxed{12}`.

#### RePPO · Member 4

correct, 759 tokens

code −0.13 verification +1.00 symbols −0.54 markdown −0.24

`\boxed{12}`

`<think>`We are asked: “What is the number of square units in the area of a triangle whose sides measure 5, 5 and 6 units?” Let me check: triangle with sides 5,5,6. This is an isosceles triangle. We can use Heron’s formula. Let  $a=5$ ,  $b=5$ ,  $c=6$ . Then semiperimeter  $s = (5+5+6)/2 = (16)/2 = 8$ . So  $\text{area} = \sqrt{s(s-a)(s-b)(s-c)} = \sqrt{8(8-5)(8-5)(8-6)} = \sqrt{8(3)(3)(2)} = \sqrt{144} = 12$ . So answer is `\boxed{12}`. But the problem said “put your final answer within `\boxed{}`”, but I must check: The problem said “put your final answer within `\boxed{}`”. So I’ll

output  $\boxed{12}$ . But let me check: Actually, check:  $833 \times 2 = 144$ ,  $\sqrt{144} = 12$ . So answer is 12.

But wait, let me check: Actually, triangle with sides 5,5,6 is valid. So answer is 12.

I must include step-by-step explanation. But instructions: "Please reason step by step, and put your final answer within  $\boxed{\{\}}\}$ ." So I'll output final answer as  $\boxed{12}$ . But instructions said "and put your final answer within  $\boxed{\{\}}\}$ ." So I'll output:  $\boxed{12}$ .

I'll check if any formatting instructions: "Make sure to put the answer (and only answer) inside  $\boxed{\{\}}\}$ ." So I'll output:  $\boxed{12}$ . But instructions said "and put your final answer within  $\boxed{\{\}}\}$ ." I'll check: The problem said "What is the number of square units in the area of a triangle whose sides measure 5, 5 and 6 units?" So answer is 12.

I'll check if any alternative method: using formula for isosceles triangle area: altitude =  $\sqrt{5^2 - (6/2)^2} = \sqrt{25 - 9} = \sqrt{16} = 4$ , area =  $1/2 \times 6 \times 4 = 12$ . So answer is 12.

I'll produce final answer:  $\boxed{12}$ .  $\text{</think>}$

We start by noting that the triangle has sides 5, 5, and 6. Since two sides are equal, it is an isosceles triangle.

Step 1: Compute the semiperimeter (s):  $s = (5 + 5 + 6) / 2 = 16 / 2 = 8$ .

Step 2: Use Heron's formula for the area (A):  $A = \sqrt{s(s-a)(s-b)(s-c)}$  Here,  $a = 5$ ,  $b = 5$ , and  $c = 6$ . So,  $A = \sqrt{8(8-5)(8-5)(8-6)} = \sqrt{8 \times 3 \times 3 \times 2} = \sqrt{144} = 12$ .

Thus, the area of the triangle is  $\boxed{12}$ .

#### Ind-PPO · Member 0

correct, 551 tokens

code -0.13 verification -0.66 symbols -0.54 markdown -0.24

$\boxed{12}$

$\text{<think>}$ We are asked: "What is the number of square units in the area of a triangle whose sides measure 5, 5 and 6 units?" We need to answer with final answer in  $\boxed{\{\}}\}$ . The answer is: Use Heron's formula. Let sides be 5, 5, 6. Then semiperimeter  $s = (5+5+6)/2 = 16/2 = 8$ . Then area =  $\sqrt{s(s-5)(s-5)(s-6)} = \sqrt{8(8-5)(8-5)(8-6)} = \sqrt{8 \times 3 \times 3 \times 2} = \sqrt{144} = 12$ . So answer is 12. But instructions: "Make sure to put the answer (and only answer) inside  $\boxed{\{\}}\}$ ." So final answer:  $\boxed{12}$ . But instructions: "Please reason step by step, and put your final answer within  $\boxed{\{\}}\}$ ." So final answer:  $\boxed{12}$ . But careful: The instructions require final answer inside  $\boxed{\{\}}\}$ . Possibly they want  $\boxed{12}$ . But instructions: "and put your final answer within  $\boxed{\{\}}\}$ ." So I'll do:  $\boxed{12}$ . But instructions: "Make sure to put the answer (and only answer) inside  $\boxed{\{\}}\}$ ." So I'll output:  $\boxed{12}$ . But instructions: "Please reason step by step, and put your final answer within  $\boxed{\{\}}\}$ ." So I'll include step-by-step explanation and then final answer inside  $\boxed{12}$ . But instructions: "and put your final answer within  $\boxed{\{\}}\}$ ." So I'll output:  $\boxed{12}$ . I'll output answer as:  $\boxed{12}$ . I'll check: "What is the number of square units in the area of a triangle whose sides measure 5, 5 and 6 units?" Yes.

I'll produce final answer:  $\boxed{12}$ . We'll produce step-by-step explanation.  $\text{</think>}$

To solve the problem, we use Heron's formula. First, we find the semiperimeter (s):

$$s = (5 + 5 + 6) / 2 = 16 / 2 = 8.$$

Next, we calculate the area (A):

$$A = \sqrt{s(s-5)(s-5)(s-6)} = \sqrt{8 \times (8-5) \times (8-5) \times (8-6)} = \sqrt{8 \times 3 \times 3 \times 2} = \sqrt{144} = 12.$$

Thus, the area of the triangle is  $\boxed{12}$ .

#### Ind-PPO · Member 1

correct, 477 tokens

code -0.13 verification -0.66 symbols -0.54 markdown -0.24

$\boxed{12}$

$\text{<think>}$ We are asked: "What is the number of square units in the area of a triangle whose sides measure 5, 5 and 6 units?" We need to produce answer inside  $\boxed{\{\}}\}$ . We need to produce step-by-step reasoning and final answer inside  $\boxed{\{\}}\}$ .

We have triangle with sides 5, 5, 6. This is an isosceles triangle with base 6 and equal sides 5. We can compute the area using Heron's formula. Let  $s = (5+5+6)/2 = 16/2 = 8$ . So area =  $\sqrt{s(s-a)(s-b)(s-c)} = \sqrt{8(8-5)(8-5)(8-6)} = \sqrt{8 \times 3 \times 3 \times 2} = \sqrt{144} = 12$ . So answer is 12.

Thus answer:  $\boxed{12}$ .

I'll produce answer:  $\boxed{12}$ . We'll produce reasoning. We'll produce answer inside  $\boxed{12}$ . We'll produce answer:  $\boxed{12}$ . We'll produce answer with step-by-step reasoning. We'll produce answer with final answer inside  $\boxed{12}$ . We'll produce answer with step-by-step reasoning.  $\text{</think>}$

To find the area of a triangle with sides 5, 5, and 6, we can use Heron's formula.

Step 1: Compute the semiperimeter ( $s$ ):  $s = (5 + 5 + 6) / 2 = 16 / 2 = 8$

Step 2: Apply Heron’s formula:  $\text{Area} = \sqrt{[s(s - a)(s - b)(s - c)]}$  Here,  $a = 5, b = 5, c = 6$ .  $\text{Area} = \sqrt{[8(8 - 5)(8 - 5)(8 - 6)]} = \sqrt{[8 \times 3 \times 3 \times 2]} = \sqrt{[144]} = 12$

Thus, the area of the triangle is  $\boxed{12}$ .

## G. Pixel-based locomotion

To test RePPO with large continuous-control policies, we train populations of 16 convolutional policies from pixels in HalfCheetah, Humanoid, and Walker2D. Observations are side views of the agent rendered on the GPU with MuJoCo Warp, converted to  $84 \times 84$  grayscale images, and stacked over the last 3 frames. Each policy is a Nature CNN encoder with separate two-layer heads of 512 units for the actor and both critics, totaling 2.5 M parameters. The tasks, descriptors, and evaluation are those of Appendix C. We train for 262 M steps in total, or 16.4 M per policy, with 16 environments per policy, rollouts of 256 steps, 4 epochs over 16 minibatches, a learning rate of  $3 \times 10^{-4}$  that decays as in Table 2 with  $S$  equal to the budget, an entropy coefficient of 0.005, value clipping,  $\ell^2 = 0.1$ , and  $K = 4$  neighbors; the remaining settings are shared with Table 2. We compare RePPO with Ind-PPO over three seeds, since archive-based baselines would need to store and train thousands of such policies.

RePPO’s populations are substantially more diverse than Ind-PPO’s in all three tasks, with Vendi Scores of 1.9–4.2 against 1.2–1.5, and achieve the higher qVS in every task (Table 12). On HalfCheetah and Walker2D, RePPO also attains the higher mean fitness, while on Humanoid it trades part of it for diversity. Figure 16 shows that RePPO’s policies spread across contact patterns, whereas Ind-PPO’s policies concentrate in a small region.

Table 12. Pixel-based locomotion, as mean  $\pm$  standard error over three seeds.

| Task        | Method  | Best fitness                     | Mean fitness                    | Vendi Score                       | qVS                               |
|-------------|---------|----------------------------------|---------------------------------|-----------------------------------|-----------------------------------|
| HalfCheetah | RePPO   | $4648 \pm 41$                    | <b><math>3257 \pm 31</math></b> | <b><math>1.90 \pm 0.03</math></b> | <b><math>6187 \pm 54</math></b>   |
|             | Ind-PPO | <b><math>5085 \pm 366</math></b> | $2887 \pm 122$                  | $1.42 \pm 0.04$                   | $4106 \pm 253$                    |
| Humanoid    | RePPO   | $2828 \pm 830$                   | $1078 \pm 63$                   | <b><math>4.22 \pm 0.10</math></b> | <b><math>4552 \pm 268</math></b>  |
|             | Ind-PPO | <b><math>3840 \pm 224</math></b> | <b><math>1709 \pm 85</math></b> | $1.47 \pm 0.05$                   | $2518 \pm 156$                    |
| Walker2D    | RePPO   | <b><math>3940 \pm 103</math></b> | <b><math>2604 \pm 72</math></b> | <b><math>3.94 \pm 0.07</math></b> | <b><math>10252 \pm 187</math></b> |
|             | Ind-PPO | $3357 \pm 311$                   | $2061 \pm 161$                  | $1.20 \pm 0.02$                   | $2461 \pm 140$                    |

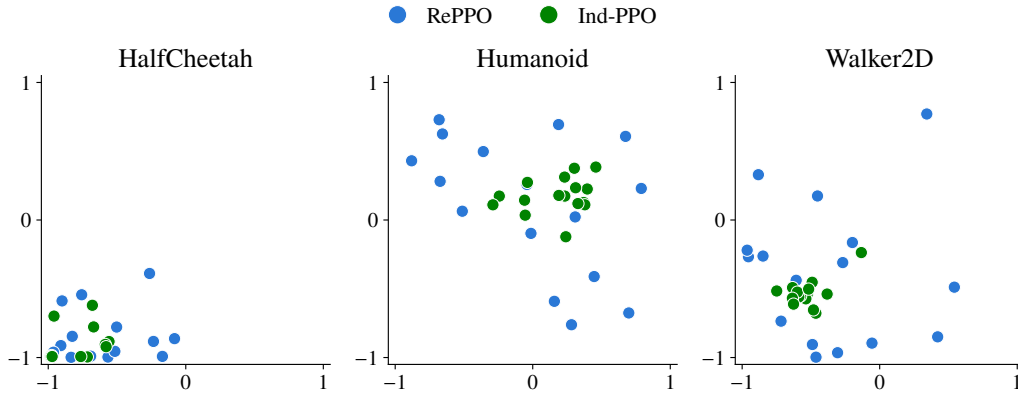


Figure 16. Behavior descriptors of pixel-based policies for one seed per task.

## H. LunarLander

Our other domains share dense behavior signals that are observed at every step. LunarLander (Towers et al., 2026) instead tests RePPO with discrete actions and a sparse, event-based descriptor. The descriptor records the lander’s horizontal position and vertical speed at its first contact with the ground, so its signal is nonzero at a single step of each episode.

Positions span the screen width, and speeds are mapped from 0 to the speed of free fall, about 2 in the environment’s units, onto  $[-1, 1]$ ; episodes without contact receive the descriptor  $(0, 0)$ . Since the reward favors fuel-efficient, gentle landings near the landing pad, diversity in this domain necessarily costs quality.

RePPO and Ind-PPO train 32 policies with 64-unit MLPs for 65.5 M steps in total, with 8 environments per policy and rollouts of 256 steps. The remaining PPO settings follow those recommended for this task by Stable-Baselines3 (Raffin et al., 2021): 4 epochs over minibatches of 64 samples, a learning rate of  $10^{-3}$ ,  $\gamma = 0.999$ ,  $\lambda = 0.98$ , and an entropy coefficient of 0.01; RePPO uses  $C_{\max} = 0.4$ . We compare against Sep-CMA-MAE, which is well suited to the small policies of this task, with a  $32 \times 32$  grid archive, 10 episodes per evaluation, and the same budget of environment steps. All methods use five seeds.

All three methods find policies that solve the task, with best fitness between 270 and 288 (Table 13). Ind-PPO’s population collapses to a single landing strategy, whereas RePPO and Sep-CMA-MAE trade mean fitness for diversity. RePPO doubles Ind-PPO’s Vendi Score and matches that of Sep-CMA-MAE with a higher mean fitness, which gives it the highest qVS. Figure 17 shows that RePPO’s policies land across the width of the screen, varying mostly in position rather than speed, whereas Ind-PPO’s policies land at nearly the same spot. RePPO’s diversity mechanism thus remains effective when the behavior signal is observed at a single event.

Table 13. LunarLander, as mean  $\pm$  standard error over five seeds.

| Method      | Best fitness                  | Mean fitness                  | Vendi Score                       | qVS                           |
|-------------|-------------------------------|-------------------------------|-----------------------------------|-------------------------------|
| RePPO       | $278 \pm 2$                   | $185 \pm 2$                   | <b><math>2.06 \pm 0.03</math></b> | <b><math>381 \pm 3</math></b> |
| Ind-PPO     | <b><math>288 \pm 1</math></b> | <b><math>272 \pm 1</math></b> | $1.05 \pm 0.01$                   | $286 \pm 1$                   |
| Sep-CMA-MAE | $270 \pm 2$                   | $135 \pm 2$                   | $2.05 \pm 0.02$                   | $276 \pm 3$                   |

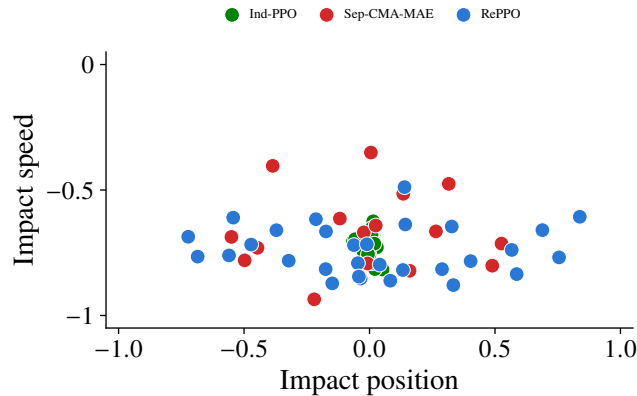


Figure 17. Behavior descriptors of LunarLander policies for one seed per method. For readability, Sep-CMA-MAE’s archive is thinned to one policy in each of 32 CVT cells, keeping the fittest.

## H.1. Computational Resources

The continuous-control, Atari, pixel-based, and LunarLander experiments were distributed over the following workstations:

- AMD Ryzen Threadripper 3990X, 64 GB RAM, 2 NVIDIA RTX 3090 GPUs;
- AMD Ryzen Threadripper 3990X, 64 GB RAM, NVIDIA RTX 4090;
- Intel Core i7-12700F, 16 GB RAM, NVIDIA RTX 3060;
- a shared server with NVIDIA RTX A6000 GPUs.

A RePPO run on Atari requires about 4 GB of GPU memory. Each LLM population was trained on five NVIDIA RTX A6000 GPUs, one member per GPU, each running both generation with vLLM (Kwon et al., 2023) and training. Training a population took 21 h for Qwen3 and 64 h for Granite, or about 100 and 320 GPU hours, and Ind-PPO required similar resources. Table 14 reports the approximated training times of locomotion experiments, in hours.

*Table 14.* Continuous control tasks’ training wall-clock time in hours. <sup>†</sup> indicates experiments run on an RTX 3090

| Method      | GPU   | Ant     | HalfCheetah | Humanoid         | Walker2D        |
|-------------|-------|---------|-------------|------------------|-----------------|
| RePPO       | 4090  | 5.1     | 8           | 5.3              | 5.7             |
| Ind-PPO     | 4090  | 4.1     | 8           | 5.3              | 5.7             |
| PPGA        | A6000 | 2.7     | 9.1         | 3.5              | 7.6             |
| PGA-ME      | 4090  | 9.8     | 26.7        | 11.6             | 22.3            |
| Sep-CMA-MAE | 4090  | 0.9–3.5 | 34          | 4.1 <sup>†</sup> | 41 <sup>†</sup> |