POPULATION-BASED POST-TRAINING FOR LARGE LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning with verifiable rewards (RLVR) is one of the most popular post-training paradigms for large language models. RLVR typically trains a single model under hyperparameters chosen before training begins, while researchers tune hyperparameters across consecutive runs. We introduce **P**opulation-**B**ased **P**ost-**T**raining (PBPT), which trains several models in parallel and lets the training process do the tuning through population-based training. PBPT periodically evaluates models on a validation set. High-performers are duplicated and their hyperparameters perturbed to explore new configurations. Low-performers stop training and are replaced by these copies. On reasoning benchmarks (Enigmata, Nemotron) and interactive tool-use environments (Gaia2, AppWorld), PBPT produces stronger agents than compute-matched groups of independent RLVR learners. Furthermore, PBPT is able to recover from poor initial hyperparameters, automatically discovering better settings over the course of training.

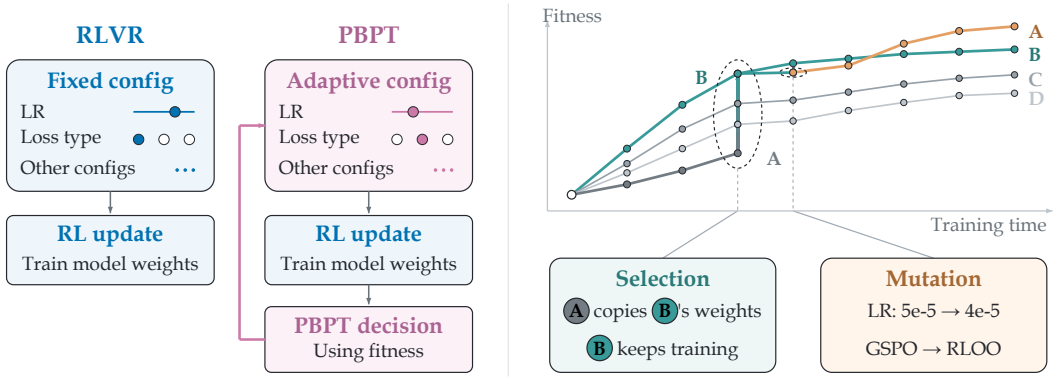
1 INTRODUCTION

Reinforcement learning with verifiable rewards (RLVR) elicits strong reasoning behavior from a pretrained large language model (LLM) without supervised reasoning traces (Guo et al., 2025). However, RLVR training can be unstable: a collapsed learner may not recover through continued training alone (Zheng et al., 2025). Moreover, training configurations are fixed before training begins, even though effective hyperparameters may change over time (Jaderberg et al., 2017). Population-based training (PBT) (Jaderberg et al., 2017) addresses both problems: it trains multiple agents concurrently, periodically copies the best-performing agents' weights to weaker peers, and perturbs their hyperparameters. As a result, a learner that stagnates is restarted from a peer that is still improving, and configurations adapt over training. Despite its success in deep reinforcement learning, PBT has seen little use in LLM post-training (Castanyer et al., 2026).

This work provides one of the first systematic studies of PBT for a population of LLM learners trained with RLVR on the same task. We introduce **PBPT** (**P**opulation-**B**ased **P**ost-**T**raining), a decentralized framework that adds an outer loop for online hyperparameter optimization to parallel RLVR training. See Figure 1 for an overview. Concretely, PBPT uses a shared filesystem as a lightweight communication medium that exposes the state of the population to all learners. Individual learners train asynchronously and periodically query this shared state. When underperforming, a learner may reinitialize from a stronger peer through a *selection* mechanism. An additional *mutation* mechanism perturbs training configurations, allowing the population to continue exploring the hyperparameter space while successful configurations propagate through selection.

We evaluate PBPT on Enigmata (Chen et al., 2025), Nemotron (NVIDIA, 2025), Gaia2 (Froger et al., 2026), and AppWorld (Trivedi et al., 2024), covering reasoning and multi-turn agentic tool use. Across all four benchmarks, PBPT outperforms independent RLVR with the same base model, number of learners, and per-learner training budget. Further experiments on Enigmata show that PBPT recovers and continues to improve from suboptimal initial RL configurations under which independent RLVR degrades substantially. We also test whether selection alone suffices, finding that PBPT's mutation mechanism is required for full recovery from suboptimal initial configurations.

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



(a) Population-based post-training **(b) PBPT decisions in a population**

Figure 1: PBPT enables parallel RLVR learners to share progress while adapting their training configurations online. Unlike independent RLVR with a fixed configuration, PBPT adds an outer optimization loop guided by validation fitness, as shown in Subfigure (a). This joint optimization is coordinated through two mechanisms, *selection* and *mutation*, as illustrated separately in Subfigure (b). Selection copies weights from the higher-fitness learner B to A. Mutation changes A's configuration while preserving its weights, allowing A to overtake B through subsequent training.

2 Related Work

Reinforcement learning with verifiable rewards. RLVR post-trains LLMs using automatically verifiable rewards and can elicit reasoning without supervised solution traces (Guo et al., 2025). GRPO is a widely used foundation for reasoning-oriented RLVR, estimating advantages from groups of sampled responses without a learned critic (Shao et al., 2024). Related objectives include REINFORCE-style estimators such as RLOO (Ahmadian et al., 2024) and sequence-level clipped objectives such as GSPO (Zheng et al., 2025). Subsequent variants target training stability, credit assignment, and data or compute efficiency, such as DAPO, VinePPO, VAPO, and PODS (Yu et al., 2025; Kazemnejad et al., 2025; Yue et al., 2025; Xu et al., 2026). PBPT complements these methods: rather than introducing a new optimization objective, it orchestrates a population of independent RLVR learners that continuously adapt their hyperparameter configurations. In this work, PBPT uses GSPO and RLOO as its underlying RL objectives, although PBPT can be implemented on top of any set of RLVR objectives, potentially switching between them during training.

Population-based training. PBT jointly optimizes model parameters and hyperparameter schedules by replacing weak learners with perturbed copies of stronger ones (Jaderberg et al., 2017). In game playing tasks, PBT has achieved human-level performance in Quake III Capture the Flag and grandmaster-level play in StarCraft II (Jaderberg et al., 2019; Vinyals et al., 2019). Recent population-oriented LLM methods train interacting policies through cooperative or collective reinforcement learning (Ma et al., 2024; Yuan et al., 2025; Zhao et al., 2026) or evolve model weights without gradient updates (Zhang et al., 2026c). These settings differ from ours because they either assign manually designed roles to population members or do not employ RLVR updates. In LLM post-training literature, PopuLoRA (Castanyer et al., 2026) is closest to our setting. However, it co-evolves asymmetric teacher and student LoRA populations for reasoning self-play rather than training homogeneous RLVR learners. Algorithmically, PBPT is closest to DexPBT, which also lets independent RL learners make selection and mutation decisions without a central coordinator (Petrenko et al., 2023). PBPT adapts this decentralized framework to the LLM post-training setting.

Prompt and harness optimization. Prompt evolution methods use evaluation or trajectory feedback to search over textual candidates while keeping model weights fixed (Yang et al., 2024; Guo et al., 2024; Fernando et al., 2024; Agrawal et al., 2026b; Tao et al., 2026; Agrawal et al., 2026a). Related methods search over executable agent designs and harnesses, including ADAS, DGM, and TTHE (Hu et al., 2025; Zhang et al., 2026a; Nie et al., 2026). Recent joint optimization methods such as E-SPL, INSPO, P^2O, and WHALE are able to optimize prompts or executable har-

nesses alongside an updating LLM policy (Zhang et al., 2026b; Zhou et al., 2025; Lu et al., 2026; Kim et al., 2026). Mutation in PBPT may similarly include non-parametric harness optimization, placing it within this broader literature, although such optimization is not the focus of this work.

Agentic control. Recent work, including Autofinetune, PostTrainBench, AutoTrainess, and Evo-Trainer, studies LLM agents that use experimental feedback to guide LLM post-training (Wei, 2026; Rank et al., 2026; Yu et al., 2026; Chen et al., 2026). PBPT also adapts training configurations based on evaluation feedback, but coordinates parallel learners through selection and hyperparameter mutation rather than asking an LLM agent to decide on the training changes.

3 PRELIMINARIES

Autoregressive language models. Let θ denote the parameters of an autoregressive language model with vocabulary $\mathcal{V}$. Given a tokenized context $\mathbf{c} \in \mathcal{V}^*$, the probability of generating a sequence $\mathbf{x} = (x_1, \ldots, x_T) \in \mathcal{V}^T$ factorizes as $p_\theta(\mathbf{x} \mid \mathbf{c}) = \prod_{t=1}^{T} p_\theta(x_t \mid \mathbf{c}, x_{1:t-1})$.

Reinforcement learning with verifiable rewards. For a context $\mathbf{c}$ from a task distribution $\mathcal{D}$, RLVR samples a response or tool-use trajectory $\mathbf{x} \sim p_\theta(\cdot \mid \mathbf{c})$ and automatically assigns it a terminal reward $R(\mathbf{c}, \mathbf{x})$. The objective is to maximize $J(\theta) = \mathbb{E}_{\mathbf{c}\sim\mathcal{D}, \mathbf{x}\sim p_\theta(\cdot|\mathbf{c})}[R(\mathbf{c}, \mathbf{x})]$. Policy gradient methods such as REINFORCE leave-one-out (RLOO) (Ahmadian et al., 2024) and group sequence policy optimization (GSPO) (Zheng et al., 2025) optimize this objective using sampled trajectories.

Population-based training. PBT maintains a population $\mathcal{P} = \{(\theta_i, h_i)\}_{i=1}^{n}$ of concurrent learners, where θ_i denotes model parameters and h_i denotes training hyperparameters (Jaderberg et al., 2017). Each member alternates between several optimization steps under its current hyperparameters and population updates based on available evaluation scores. Three terms are useful for describing this process: *fitness*, *selection*, and *mutation*.

Fitness. The fitness $f_i = f(\theta_i)$ of member i is a scalar evaluation score for its current checkpoint, with higher values indicating better checkpoints. Unlike the RLVR reward $R(\mathbf{c}, \mathbf{x})$ for an individual trajectory, fitness compares checkpoints during population updates.

Selection. Selection may replace a lower-fitness member i with a higher-fitness member j, either after comparing the two directly or by replacing a fixed fraction of the lowest-fitness members. We call member i the recipient and member j the donor. The recipient copies the donor's model parameters, optimizer state, and typically its hyperparameters.

Mutation. Mutation changes a member's hyperparameters during training according to a mutation operator, written $h_i \leftarrow \mathrm{Mutate}(h_i)$. In the original formulation of PBT (Jaderberg et al., 2017), a hyperparameter is either multiplied by 0.8 or 1.2, or resampled from its initial prior distribution. The member then resumes optimization under the mutated configuration.

4 POPULATION-BASED POST-TRAINING

We introduce **PBPT** (**P**opulation-**B**ased **P**ost-**T**raining), a framework that applies PBT to LLM RLVR post-training. PBPT lets underperforming learners inherit stronger peers' learned state while exploring training configurations through hyperparameter mutation.

4.1 PBPT OVERVIEW

PBPT trains a fixed population of n members in parallel. Each member samples responses or tool-use trajectories, called *rollouts*, and updates its policy through RLVR, with periodic validation evaluations. Each evaluation triggers a *population decision* that may change the member's training state.

We denote the training state of member i by $\mathrm{state}_i = (\theta_i, \omega_i, h_i)$, where θ_i are its policy parameters, ω_i is its optimizer state, and h_i is its hyperparameter configuration. For RLVR, this configuration includes choices such as the learning rate, the number of rollouts per task, and the RL loss family. Let k index the local validation evaluations of member i. At evaluation k, the member forms $\mathrm{record}_{i,k} =$

Algorithm 1. Population-Based Post-Training (PBPT)

Input: Initial population $\{\text{state}_i\}_{i=1}^{n}$, training and validation splits, and a per-member budget
Output: Trained population $\{\text{state}_i\}_{i=1}^{n}$

Let the shared record set $\mathcal{S} \leftarrow \varnothing$
for each member $i \in \{1, \ldots, n\}$ asynchronously in parallel **do**
  while member i has remaining training budget **do**
    Let $\text{state}_i \leftarrow \textsc{Train}(\text{state}_i)$ ▷ RLVR until the next evaluation
    Let $\text{fitness}_i \leftarrow \textsc{Eval}(\text{state}_i)$ ▷ Validation fitness
    Publish $(\text{state}_i, \text{fitness}_i, \text{time}_i)$ to $\mathcal{S}$
    Let $\mathcal{C}_i$ contain the current record and one compute-matched record per active peer
    Let $(\text{action}, \text{donor}) \leftarrow \textsc{Decide}(i, \mathcal{C}_i, \mathcal{S})$
    if $\text{action} = \textsc{Select-and-Mutate}$ **then** ▷ Copy donor weights, then mutate
      Let $(\theta, \omega, h) \leftarrow \text{state}_i$
      Let $(\theta_{\text{donor}}, \omega_{\text{donor}}, h_{\text{donor}}) \leftarrow$ the donor's state
      Let $\text{state}_i \leftarrow (\theta_{\text{donor}}, \omega_{\text{donor}}, \textsc{Crossover}(h, h_{\text{donor}}))$
      Let $\text{state}_i \leftarrow \textsc{Mutate}(\text{state}_i)$
    else if $\text{action} = \textsc{Mutate}$ **then** ▷ Keep weights, mutate configuration
      Let $\text{state}_i \leftarrow \textsc{Mutate}(\text{state}_i)$
return $\{\text{state}_i\}_{i=1}^{n}$

$(\text{state}_{i,k}, \text{fitness}_{i,k}, \text{time}_{i,k})$, where $\text{fitness}_{i,k}$ is the validation score and $\text{time}_{i,k}$ measures compute as accumulated job runtime. We will omit the evaluation index k in the algorithms.

Then, member i publishes $\text{record}_{i,k}$ to a shared record set $\mathcal{S}$ and constructs a comparison set $\mathcal{C}_i$ from the records in $\mathcal{S}$. Based on $\mathcal{C}_i$, PBPT uses DECIDE to choose one of three actions: KEEP leaves the current state unchanged, MUTATE applies mutation to the current configuration, and SELECT-AND-MUTATE applies selection to inherit the learned state of a higher-fitness checkpoint, followed by mutation. The member then continues RLVR from the resulting state until its next evaluation.

Algorithm 1 summarizes the asynchronous training loop, and Algorithm 2 in Section A.3 defines DECIDE, CROSSOVER, and MUTATE. The remaining subsections describe asynchronous execution and compute-matched peer comparisons, selection and state transfer, and configuration mutation.

4.2 Asynchronous Execution with Compute-Matched Peer Comparisons

Each member makes its own population decisions using evaluation records in shared storage, without a central coordinator. This allows each individual member to continue training even when others start late, progress slowly, or are temporarily unavailable.

Note that asynchronous members may reach their evaluations after different amounts of training compute. Comparing only their latest checkpoints can therefore favor members that have trained longer. PBPT instead matches records by accumulated compute: a peer record is *eligible* for member i's comparison only if its recorded compute does not exceed $\text{time}_{i,k}$.

For each active peer, member i retrieves the eligible record with the greatest accumulated compute. These peer records and member i's current record form the comparison set $\mathcal{C}_i$. Peers that have not made recent progress are excluded using the recency checks in Section A.3.

4.3 Selection and State Transfer

PBPT makes population decisions using a decentralized rank-based rule similar to DexPBT (Petrenko et al., 2023). Member i ranks the members represented in $\mathcal{C}_i$ by fitness, placing the highest-fitness fraction in the *winner set* $\mathcal{W}$ and the lowest-fitness fraction in the *loser set* $\mathcal{L}$. Members outside $\mathcal{L}$ continue training unchanged. A member in $\mathcal{L}$ is eligible for selection or mutation if it has made sufficient local training progress and there are enough peers for comparison.

For eligible members, PBPT compares the fitness gap to the best-performing member in $\mathcal{C}_i$ against two adaptive thresholds. A small performance gap results in KEEP, a larger gap triggers MUTATE,

and sufficiently severe underperformance triggers SELECT-AND-MUTATE. Both thresholds depend on the fitness spread and the best fitness in $\mathcal{C}_i$ (see Section A.3 for details).

For SELECT-AND-MUTATE, PBPT forms a donor pool from the winner records in $\mathcal{C}_i$ and selected high-fitness historical records in $\mathcal{S}$. These historical records must have fitness at least the best in $\mathcal{C}_i$ and compute no greater than $\mathrm{time}_{i,k}$ (Section A.3). PBPT samples a donor record uniformly from the combined pool. The recipient copies the donor record's policy parameters and optimizer state. CROSSOVER independently chooses each hyperparameter from the recipient or donor configuration with equal probability. The recipient then mutates the combined configuration and resumes training.

4.4 CONFIGURATION MUTATION

PBPT represents a configuration of d mutable hyperparameters as $h = (h_1, \ldots, h_d) \in \mathcal{H} = \mathcal{H}_1 \times \cdots \times \mathcal{H}_d$. Each hyperparameter h_q has a declared type and a valid range $\mathcal{H}_q$. Our search spaces include settings for optimization, rollout collection, task sampling, and scenario replay, which resamples previously encountered tasks. Section A.2 lists all hyperparameters and their ranges.

The MUTATE operator leaves policy parameters θ and optimizer state ω unchanged while independently perturbing each hyperparameter in h with probability p_{mut}. Numeric mutations use random scaling, followed by rounding and clamping as needed. Boolean values are flipped, and categorical values are resampled from their allowed choices. Section A.3 gives the exact mutation rules, including range and cross-parameter constraints.

Both MUTATE and SELECT-AND-MUTATE use this mutation operator. Before training, we also mutate a subset of members to diversify their initial configurations.

5 EXPERIMENTS

In this section, we conduct experiments to evaluate the effectiveness of PBPT in LLM post-training. We focus on reasoning and agentic tool-use tasks, using compute-matched independent RLVR runs as the main baseline. We also show that PBPT adapts from suboptimal initial RL configurations and use ablations to establish that mutation is needed for PBPT to achieve its full gains.

Benchmarks. We consider four benchmarks spanning reasoning and agentic tool use: Enigmata, Nemotron, Gaia2, and AppWorld.

Enigmata (Chen et al., 2025) consists of 36 puzzle families covering constraint satisfaction, search, symbolic reasoning, and logic. We train on a cleaned version of the benchmark, correcting invalid problems and filtering out selected malformed task categories; see Section A.1 for details. We generate training instances using the official script, hold out a shuffled subset for validation, and sample the test set from Enigmata-Eval. After this cleanup, the resulting splits contain 161,213 training, 229 validation, and 998 test instances across 35 families.

Nemotron (NVIDIA, 2025) refers to NVIDIA's Nemotron-3-Nano RL training blend. We use 6 task subsets covering STEM multiple-choice questions, competition mathematics, instruction following, structured JSON generation, and competitive programming. We split the data into disjoint sets, using 81,725 tasks for training and 512 tasks each for validation and test.

Gaia2 (Froger et al., 2026) evaluates multi-turn agents on personal-assistant tasks involving applications such as email, calendar, contacts, and files. We use its 800 public scenarios across five capability categories: execution, search, adaptability, time, and ambiguity. We split the data by *universe*, an application environment with its own user data, to keep the underlying application data disjoint across splits. This gives 458 training, 153 validation, and 189 test scenarios.

AppWorld (Trivedi et al., 2024) evaluates interactive coding agents that complete everyday tasks by writing and executing code against application APIs. We use the 90 training tasks and 55 of the 57 development tasks for validation, excluding 2 development tasks with bugs in their definitions. We evaluate on 168 normal and 417 challenge test tasks.

Algorithms. We compare a PBPT population of 8 learners with 8 independent RLVR runs. Both methods start from the same base model and use the same per-learner training resources. Learners

use different random seeds and share an RL configuration before PBPT's initial mutation. PBPT may also change this configuration during training. Section A.1 describes the training rewards, and Section A.2 gives the initial RL configurations and PBPT search spaces.

Evaluation. We use the training split for RLVR updates, the validation split for population decisions, and the test split only for reporting. We report accuracy on Enigmata and Nemotron and task goal completion on Gaia2 and AppWorld, defined as the fraction of tasks completed successfully. Success requires all task checks to pass. Section A.1 provides data and evaluation details.

Implementation. We use Qwen3.5-4B for Enigmata, Nemotron, and AppWorld, and Qwen3.5-9B for Gaia2. We use low-rank adaptation (LoRA) (Hu et al., 2022) with rank 8 and scaling factor 16. We use Accelerate (Gugger et al., 2022) for distributed training and vLLM (Kwon et al., 2023) for rollout generation. Each learner uses 4 NVIDIA B200 GPUs for training, rollout generation, and evaluation. We compare the methods over the first 66 hours of training per learner on Enigmata, Nemotron, and AppWorld, and 90 hours on Gaia2. We measure training time as each learner's accumulated training-iteration time, including rollout collection and policy updates but excluding startup and intervals when training is stopped (see Section A.3 for additional details).

5.1 Performance Against Compute-Matched Independent RLVR

We present the training curves of PBPT and independent RLVR in Figure 2. Across all four benchmarks, PBPT achieves higher mean validation and test performance than independent RLVR under matched training budgets. PBPT maintains its advantage through the later stages of training.

The gains extend from validation to the held-out test splits, and the test splits are not used for population decisions. Therefore, coordinating parallel learners with PBPT can improve RLVR post-training across reasoning and multi-turn tool-use tasks under matched training budgets.

5.2 Adaptation from Suboptimal RL Configurations

RL hyperparameters can be difficult to tune. Seemingly reasonable choices may lead to overly aggressive updates or unstable training (Yu et al., 2025; Zheng et al., 2025). We therefore evaluate whether PBPT can adapt from a plausible but suboptimal initial configuration. On Enigmata, we manually choose values within PBPT's search ranges, including a higher learning rate, wider clipping bounds, more optimization epochs per iteration, and a smaller minibatch than in the well-tuned initial configuration (Section A.2.2). We compare 8 independent RLVR runs with 8 PBPT learners, all starting from this configuration with the same base model and per-learner resources.

Figure 3 shows that independent RLVR initially improves but then degrades substantially as training continues and never recovers. In contrast, PBPT continues to improve, maintaining its gains through the later training. These results show that PBPT can sustain policy improvement by searching for effective configurations during training, even when starting from a suboptimal initial configuration.

5.3 The Role of Mutation

PBPT combines selection with hyperparameter mutation. To isolate the contribution of mutation, we evaluate a selection-only variant, *PBPT without mutation*, which disables mutation both at initialization and throughout training.

We compare full PBPT, PBPT without mutation, and independent RLVR on Enigmata under both the tuned initial configuration from Section 5.1 and the suboptimal initial configuration from Section 5.2. Each method uses 8 learners with the same base model and per-learner resources.

As shown in Figure 4, the effect of mutation depends on the initial configuration. With the tuned configuration, selection-only PBPT performs comparably to full PBPT, and both outperform independent RLVR. With the suboptimal configuration, selection alone also prevents the large degradation observed in independent RLVR, but reaches a lower test accuracy than full PBPT. We therefore see that mutation is necessary for PBPT's full gains from suboptimal initial configurations.

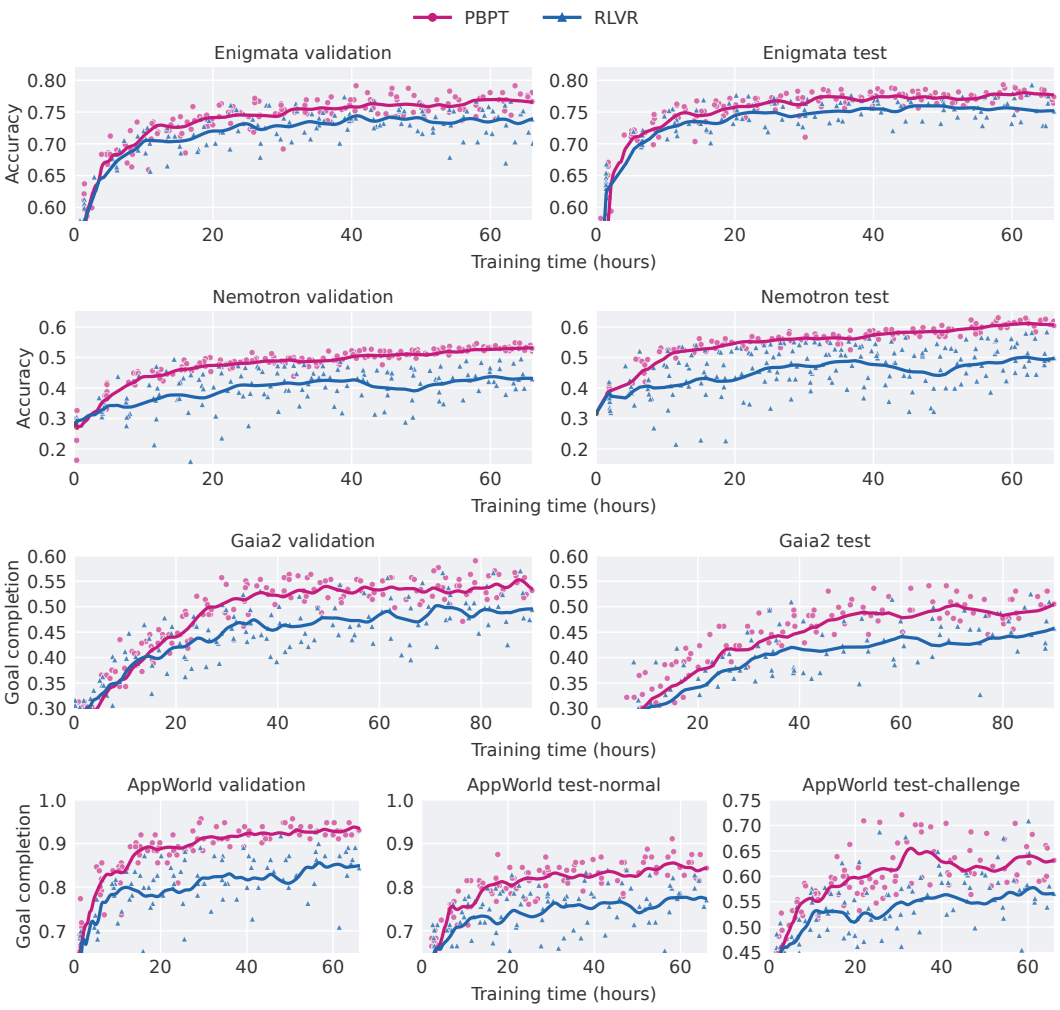


Figure 2: Comparison of PBPT and independent RLVR on Enigmata, Nemotron, Gaia2, and AppWorld. The x-axis shows accumulated training time per learner. The y-axis shows accuracy on Enigmata and Nemotron, and task goal completion on Gaia2 and AppWorld. Scatter points show subsampled evaluation results, and solid lines show smoothed means of the latest available evaluation from each population member or independent seed. The shared point at zero training time reports a separate evaluation of the base model. Compared with independent RLVR, PBPT achieves higher mean validation and test performance under matched training budgets.

5.4 Population-Level and Individual Performance

We summarize in Table 1 the Enigmata runs with both tuned and suboptimal initial configurations over the final 20% of the training budget. We first average each member's validation and test evaluations within this window. We then report the mean test accuracy across all 8 members and the test means of the members with the highest and lowest validation means.

As shown in Table 1, PBPT improves mean test accuracy and the test performance of both the highest- and lowest-ranked members under both initial configurations. Notably, the gains under the tuned initialization are obtained on Enigmata, where the independent RLVR baseline is already particularly strong and carefully tuned, leaving relatively little headroom for improvement. We observe that the gap between PBPT and RLVR is larger on the other environments.

Under the tuned initialization, PBPT still improves all three measures. The smaller gap for the highest-ranked member reflects the fact that an occasional well-behaved RLVR trajectory can already reach high performance when initialized from a well-tuned configuration. PBPT's advantage

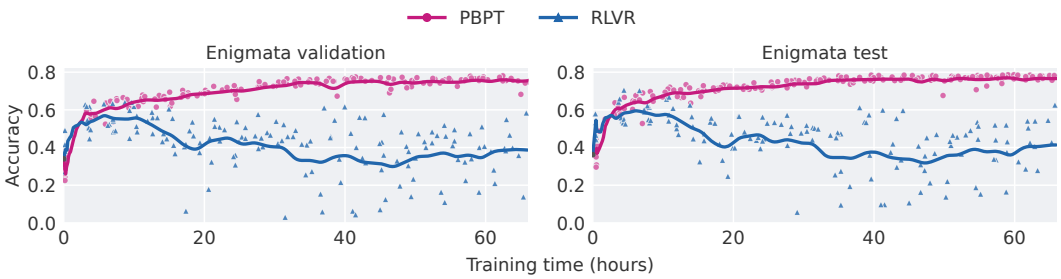


Figure 3: Comparison of PBPT and independent RLVR on Enigmata with a suboptimal initial configuration. The axes, points, and lines follow the same convention as in Figure 2. Over the course of training, PBPT continues to improve while independent RLVR loses its early gains.

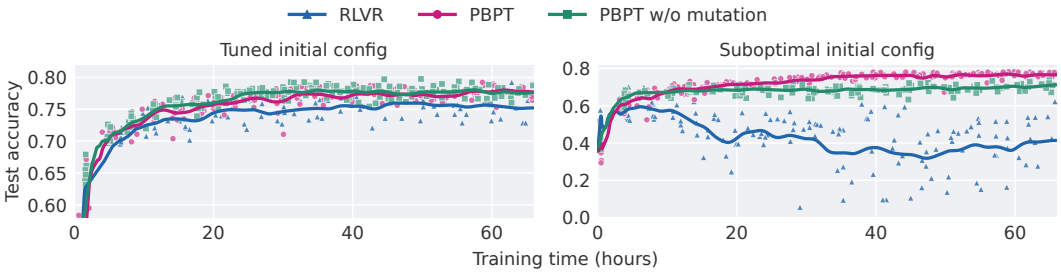


Figure 4: Effect of mutation on Enigmata test performance under tuned and suboptimal initial configurations. The axes, points, and lines follow the same convention as in Figure 2. PBPT without mutation underperforms full PBPT under the suboptimal configuration, but performs comparably under the tuned configuration. Both PBPT variants outperform independent RLVR.

is therefore most visible in the population as a whole: it raises average performance and improves the lower end of the population while retaining strong top-performing members.

The advantage becomes much larger under the suboptimal initialization. Here, PBPT substantially outperforms independent RLVR across all three measures, showing that mutation enables the population to move away from a poor initial configuration while selection propagates successful learners.

5.5 PBPT Behavior Analysis

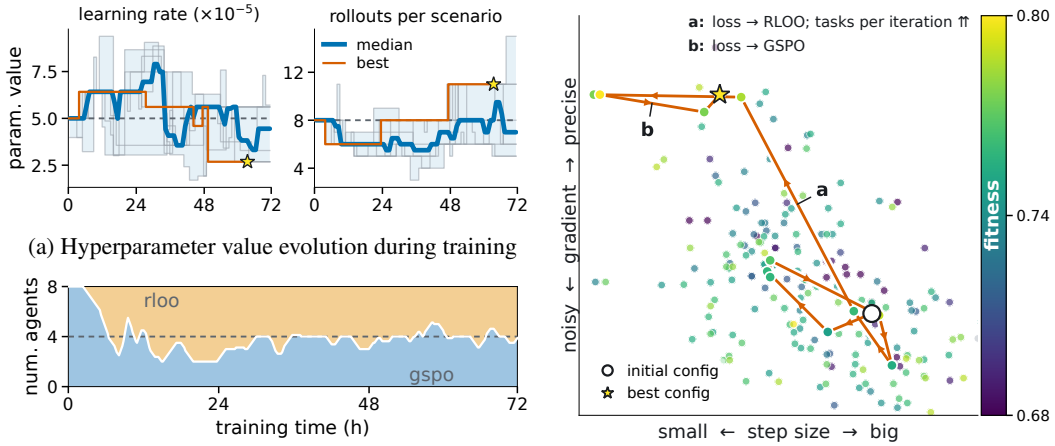
Figure 5 shows how PBPT explores and adapts training configurations over the course of the Enigmata run from Figure 2. Figure 5a shows emergent schedules for selected hyperparameters: the learning rate rises early and then decreases, while the number of rollouts per scenario tends to increase later in training. The population also continues to explore both GSPO and RLOO throughout training rather than converging to a single loss family (Figure 5b).

To visualize PBPT's hyperparameter search, Figure 5c maps configurations onto two axes formed from weighted combinations of hyperparameters (Section A.2.4 and Table 6). The horizontal axis captures optimization aggressiveness and increases with factors such as larger learning rates and looser GSPO clipping. The vertical axis captures the tradeoff between cheaper, noisier updates and more precise, expensive gradient estimates, increasing with larger batch sizes, more rollouts per scenario, and more on-policy updates as in RLOO. The best lineage follows a highly non-monotonic path through this space, repeatedly moving between distinct training regimes through mutation.

Overall, PBPT tends to favor larger optimization steps and cheaper, noisier gradients earlier in training for faster wall-time iteration. It shifts toward more conservative, lower-variance, and more on-policy updates later in training, when useful learning signal becomes sparser.

Table 1: Enigmata test accuracy (%) under tuned and suboptimal initial RL configurations, averaged over the evaluation results in the final 20% of the training budget. Avg. test weights all 8 members equally. Best-val. test and Worst-val. test report the test means of the members with the highest and lowest validation means, respectively. Bold marks the highest value in each column.

Method	Tuned initial configuration			Suboptimal initial configuration		
	Avg. test	Best-val. test	Worst-val. test	Avg. test	Best-val. test	Worst-val. test
RLVR	75.40	78.00	73.03	40.87	45.86	26.70
PBPT w/o mutation	**77.73**	77.76	**77.54**	70.07	70.47	69.68
PBPT	77.59	**78.12**	77.03	**76.40**	**76.74**	**75.79**



(a) Hyperparameter value evolution during training

(b) Number of members using each RL loss

(c) Hyperparameter path to the best checkpoint

Figure 5: Evolution of the training configurations in the PBPT population on Enigmata. Subfigure (a) shows the learning rate and rollouts per scenario for individual members in gray, their range as a shaded band, and the population median in blue. Dashed lines mark initial values. Subfigure (b) shows the number of members using each RL loss. Subfigure (c) maps explored configurations to two interpretable axes, with points colored by the best fitness recorded under each configuration. The orange paths in Subfigures (a) and (c) trace the training history leading to the checkpoint with the highest fitness. In Subfigure (c), the circle and star mark the initial and best configurations.

6 Discussion and Conclusion

We introduced PBPT, a decentralized framework that integrates online hyperparameter optimization with parallel RLVR post-training by mutating hyperparameters and transferring learned weights from stronger to weaker learners. We demonstrated that, on reasoning and tool use benchmarks, PBPT produces stronger agents than compute-matched independent RLVR. Additionally, PBPT automatically discovers better configurations and recovers from suboptimal ones during training.

While our experiments focused on PBPT with *parametric mutations* (i.e., mutations of numeric hyperparameters and discrete training choices), the PBPT framework can also accommodate *non-parametric mutations*, such as changes to an agent's system prompt or execution harness. In this setup, population decisions, selections, and mutations can be made by an *LLM agent* rather than fixed rules. This is closely related to autonomous experimentation systems such as Autoresearch (Karpathy, 2026), which uses an LLM agent to propose and evaluate changes to training code. Our preliminary experiments with this extension suggest that current systems remain fragile and highly sensitive to prompting, agent choice, and the representation of candidate modifications. As agents improve in reliability and long-horizon reasoning, integrating them into PBPT could become a powerful avenue for more autonomous adaptation and an exciting direction for future work.

AI USE STATEMENT

We used generative AI tools to assist with manuscript drafting and editing, literature search, code implementation and debugging, figure preparation, and plotting of experimental results. The core research ideas and algorithmic design were developed by the human authors. The authors are responsible for verifying AI-assisted material and for the manuscript, code, and results.

ETHICS STATEMENT

This work studies LLM post-training on reasoning and tool-use benchmarks. We do not identify ethical concerns specific to PBPT beyond those associated with training and deploying LLMs.

REPRODUCIBILITY STATEMENT

To support reproducibility, we describe PBPT in Section 4 and provide pseudocode in Algorithms 1 and 2. Section 5 specifies the benchmarks, data splits, models, evaluation protocol, and training resources. Section A provides further details on benchmark modifications, initial RL configurations, reward functions, PBPT search spaces, and implementation.

REFERENCES

Lakshya A. Agrawal, Donghyun Lee, Shangyin Tan, Wenjie Ma, Karim Elmaaroufi, Sanjit A. Seshia, Koushik Sen, Dan Klein, Ion Stoica, Joseph E. Gonzalez, et al. optimize_anything: A universal API for optimizing any text parameter. In *ACM Conference on AI and Agentic Systems*, 2026a.

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, et al. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, 2026b.

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting REINFORCE-style optimization for learning from human feedback in LLMs. In *Annual Meeting of the Association for Computational Linguistics*, 2024.

Roger Creus Castanyer, Geoffrey Bradway, Lorenz Wolf, Maxwill Lin, Augustine N. Mavor-Parker, and Matthew J. Sargent. PopuLoRA: Co-evolving LLM populations for reasoning self-play. arXiv preprint arXiv:2605.16727, 2026.

Guhong Chen, Yingcheng Shi, Yongbin Li, Binhua Li, Xander Xu, Hu Wei, Shiwen Ni, Min Yang, and Jieping Ye. EvoTrainer: Co-evolving LLM policies and training harnesses for autonomous agentic reinforcement learning. arXiv preprint arXiv:2606.03108, 2026.

Jiangjie Chen, Qianyu He, Siyu Yuan, Aili Chen, Zhicheng Cai, Weinan Dai, Hongli Yu, Jiaze Chen, Xuefeng Li, Qiying Yu, et al. Enigmata: Scaling logical reasoning in large language models with synthetic verifiable puzzles. In *Advances in Neural Information Processing Systems*, 2025.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. In *International Conference on Machine Learning*, 2024.

Romain Froger, Pierre Andrews, Matteo Bettini, Amar Budhiraja, Ricardo Silveira Cabral, Virginie Do, Emilien Garreau, Jean-Baptiste Gaya, Hugo Laurençon, Maxime Lecanu, et al. Gaia2: Benchmarking LLM agents on dynamic and asynchronous environments. In *International Conference on Learning Representations*, 2026.

Sylvain Gugger, Lysandre Debut, Thomas Wolf, Philipp Schmid, Zachary Mueller, Sourab Mangrulkar, Marc Sun, and Benjamin Bossan. Accelerate: Training and inference at scale made simple, efficient and adaptable. `https://github.com/huggingface/accelerate`, 2022.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *International Conference on Learning Representations*, 2024.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, 2025.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. Population based training of neural networks. arXiv preprint arXiv:1711.09846, 2017.

Max Jaderberg, Wojciech M. Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castañeda, Charles Beattie, Neil C. Rabinowitz, Ari S. Morcos, Avraham Ruderman, et al. Human-level performance in 3D multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019.

Andrej Karpathy. Autoresearch: AI agents running research on single-GPU nanochat training automatically. `https://github.com/karpathy/autoresearch`, 2026. GitHub repository.

Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordoni, Siva Reddy, Aaron C. Courville, and Nicolas Le Roux. VinePPO: Refining credit assignment in RL training of LLMs. In *International Conference on Machine Learning*, 2025.

Haechan Kim, Yoonho Lee, Gisang Lee, Chelsea Finn, and Kangwook Lee. WHALE: A simple recipe for joint harness-weight optimization. arXiv preprint arXiv:2609.00196, 2026.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *ACM Symposium on Operating Systems Principles*, 2023.

Xinyu Lu, Kaiqi Zhang, Jinglin Yang, Boxi Cao, Yaojie Lu, Hongyu Lin, Min He, Xianpei Han, and Le Sun. P^2O: Joint policy and prompt optimization. arXiv preprint arXiv:2603.21877, 2026.

Hao Ma, Tianyi Hu, Zhiqiang Pu, Boyin Liu, Xiaolin Ai, Yanyan Liang, and Min Chen. Coevolving with the other you: Fine-tuning LLM with sequential cooperative multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, 2024.

Jun Nie, Yonggang Zhang, Jun Song, Qianshu Cai, Dahai Yu, Yike Guo, Xinmei Tian, and Bo Han. TTHE: Test-time harness evolution. arXiv preprint arXiv:2607.08124, 2026.

NVIDIA. Nemotron 3 Nano: Open, efficient mixture-of-experts hybrid Mamba-Transformer model for agentic reasoning. arXiv preprint arXiv:2512.20848, 2025.

Aleksei Petrenko, Arthur Allshire, Gavriel State, Ankur Handa, and Viktor Makoviychuk. DexPBT: Scaling up dexterous manipulation for hand-arm systems with population based training. In *Robotics: Science and Systems*, 2023.

Aleksei Petrenko, Ben Lipkin, Kevin Chen, Erik Wijmans, Marco Cusumano-Towner, Raja Giryes, and Philipp Krähenbühl. Entropy-preserving reinforcement learning. In *International Conference on Learning Representations*, 2026.

Ben Rank, Hardik Bhatnagar, Ameya Prabhu, Shira Eisenberg, Karina Nguyen, Matthias Bethge, and Maksym Andriushchenko. PostTrainBench: Can LLM agents automate LLM post-training? arXiv preprint arXiv:2603.08640, 2026.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. arXiv preprint arXiv:2402.03300, 2024.

Wangcheng Tao, Han Wu, and Weng-Fai Wong. SePO: Self-evolving prompt agent for system prompt optimization. arXiv preprint arXiv:2606.04465, 2026.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Annual Meeting of the Association for Computational Linguistics*, 2024.

Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

Wei Wei. Autonomous llm post-training with tunix on tpus. Google Developers Blog, September 2026. URL `https://developers.googleblog.com/en/autonomous-llm-post-training-with-tunix-on-tpus/`. Accessed: 2026-09-15.

Yixuan Even Xu, Yash Savani, Fei Fang, and J. Zico Kolter. Not all rollouts are useful: Down-sampling rollouts in LLM reinforcement learning. *Transactions on Machine Learning Research*, 2026.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations*, 2024.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, et al. DAPO: An open-source LLM reinforcement learning system at scale. In *Advances in Neural Information Processing Systems*, 2025.

Zhaojian Yu, Penghao Yin, Shuzheng Gao, Shilin He, Kai Cai, and Xiao-Ping Zhang. AutoTrainess: Teaching language models to improve language models autonomously. arXiv preprint arXiv:2606.31551, 2026.

Wenzhen Yuan, Shengji Tang, Weihao Lin, Jiacheng Ruan, Ganqu Cui, Bo Zhang, Tao Chen, Ting Liu, Yuzhuo Fu, Peng Ye, et al. Wisdom of the crowd: Reinforcement learning from coevolutionary collective feedback. arXiv preprint arXiv:2508.12338, 2025.

Yu Yue, Yufeng Yuan, Qiying Yu, Xiaochen Zuo, Ruofei Zhu, Wenyuan Xu, Jiaze Chen, Chengyi Wang, TianTian Fan, Zhengyin Du, et al. VAPO: Efficient and reliable reinforcement learning for advanced reasoning tasks. arXiv preprint arXiv:2504.05118, 2025.

Jenny Zhang, Shengran Hu, Cong Lu, Robert T. Lange, and Jeff Clune. Darwin Gödel Machine: Open-ended evolution of self-improving agents. In *International Conference on Learning Representations*, 2026a.

Lunjun Zhang, Ryan Chen, and Bradly C. Stadie. Evolutionary system prompt learning for reinforcement learning in LLMs. arXiv preprint arXiv:2602.14697, 2026b.

Yiqun Zhang, Peng Ye, Xiaocui Yang, Shi Feng, Shufei Zhang, Lei Bai, Wanli Ouyang, and Shuyue Hu. Nature-inspired population-based evolution of large language models. In *Annual Meeting of the Association for Computational Linguistics*, 2026c.

Yujie Zhao, Lanxiang Hu, Yang Wang, Minmin Hou, Hao Zhang, Ke Ding, and Jishen Zhao. Stronger-MAS: Multi-agent reinforcement learning for collaborative LLMs. In *International Conference on Learning Representations*, 2026.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, et al. Group sequence policy optimization. arXiv preprint arXiv:2507.18071, 2025.

Han Zhou, Xingchen Wan, Ivan Vulić, and Anna Korhonen. Agentic policy optimization via instruction-policy co-evolution. arXiv preprint arXiv:2512.01945, 2025.

A Experimental Details

A.1 Benchmarks and Evaluation

We describe each benchmark's data, evaluation, and rewards, using the term *scenario* to denote a task instance. The decoding settings below are the initial values used by each method. Training rollouts are always sampled at temperature 1, without top-*p*, top-*k*, or min-*p* filtering.

PBPT may mutate the evaluation temperature, which does not affect training rollouts. On Gaia2 and AppWorld, the per-message generation limit is also mutable and applies to both training rollouts and evaluation. Validation and test use each member's current settings, whereas independent RLVR keeps these settings fixed.

Evaluation schedule. Each learner saves one checkpoint per training iteration. Evaluation proceeds in *passes*, and each pass selects one checkpoint. The selected checkpoint is the newest finished checkpoint that is at least 1 and at most 20 iterations (Enigmata, Nemotron) or 10 iterations (Gaia2, AppWorld) after the previously selected one. Checkpoints are therefore skipped when training outpaces evaluation. A pass evaluates the selected checkpoint on the validation split if the pass index is a multiple of p_{val} and on test if it is a multiple of p_{test}. A pass on which neither split is due evaluates nothing, and its checkpoint is skipped. The spacing between evaluated checkpoints therefore depends on evaluation speed, and we report measured spacings for each benchmark below.

Compute. In all experiments, each PBPT member and each independent RLVR seed runs on one node with 4 B200 GPUs. The learner measures its peak memory use and gives the remaining GPU memory to the vLLM inference servers. Evaluation runs concurrently as a separate process on the same node, using these inference servers. The same GPUs host policy updates, rollout generation, evaluation, and, on Gaia2, the LLM judges, with no separate evaluation allocations. Concurrent evaluation slows training iterations and therefore contributes to the training-time clock.

For the 8-learner comparisons, each method has a nominal training budget of $8 \times 4 \times 66 = 2{,}112$ GPU-hours on Enigmata, Nemotron, and AppWorld, and $8 \times 4 \times 90 = 2{,}880$ GPU-hours on Gaia2. These budgets exclude overhead outside the training-time clock, such as cluster queueing and preemptions.

Curve aggregation and smoothing. For each metric, we first construct a population mean over evaluation time. At each evaluation time appearing in any learner's trajectory, we average the most recent available score from every learner that has already been evaluated. If a learner stops early, its final score continues to contribute to the mean.

For visualization (see Figures 2 to 4), we smooth the resulting mean curve with a Gaussian kernel that uses observations both before and after each plotted time. At training time t hours, the kernel standard deviation is $0.75 \min\{1, \max\{0.05, t/12\}\}$ hours. Scatter points are subsampled only for readability, targeting 150 points per method and panel (100 on AppWorld); this does not affect the mean curves. Table 1 instead reports unsmoothed evaluations within its stated averaging window.

A.1.1 Enigmata

Data. Enigmata (Chen et al., 2025) contains 36 puzzle families across 7 reasoning categories, including Minesweeper, Sudoku, and Countdown. In this paper, we train and evaluate on a significantly modified version of Enigmata which includes multiple bug fixes (see "Benchmark corrections and improvements" below).

Each task requires a single response without tool use. We generate instances using the official Enigmata script and construct the validation set by shuffling the generated data and holding out a subset. The remaining instances form the training set, and the test set is sampled from Enigmata-Eval. After the corrections described below, the splits comprise 161,213 training, 229 validation, and 998 test instances over 35 families. Task identifiers are unique across the pooled splits. We also checked task families for overlap between training and evaluation. In the crossword family, 84.8% of evaluation clue–answer pairs appeared verbatim in training, contributing to its removal. For the regenerated fifteen-puzzle boards, no evaluation board matches or is one move away from a training board.

Evaluation. We sample 2 rollouts per validation task and 1 per test task. A rule-based verifier scores each response as 1 if correct and 0 otherwise, and we report the mean over all rollouts. The initial evaluation temperature is 0.6, with top-*p* set to 0.95 and top-*k* to 20. We do not use multi-token prediction. We allow 48,000 context tokens and up to 24,576 generated tokens per response. Both methods use $p_{\text{val}} = 2$. In the main comparison, PBPT uses $p_{\text{test}} = 4$ and independent RLVR uses $p_{\text{test}} = 6$. The baseline's evaluation passes also include additional held-out splits not reported here. In the reported runs, consecutive validated checkpoints are a median of 2 training iterations (0.4 hours) apart for PBPT and 4 iterations (0.55 hours) for independent RLVR. Consecutive test evaluations are a median of 5 iterations (0.8 hours) apart for PBPT and 14 iterations (2.3 hours) for independent RLVR. In the suboptimal-configuration experiments, all three methods use $p_{\text{test}} = 4$, and validated checkpoints are a median of 2 to 3 iterations apart.

Reward. The task-specific verifier checks the extracted answer against the reference solution or the puzzle's constraints. The training reward is 1 for a correct response and 0 otherwise, with no length, format, or other penalty terms. We score only the text after the final `</think>` tag, which marks the end of the reasoning trace. A response with an unclosed reasoning trace scores 0. A missing or malformed answer is rejected by the verifier and scores 0, as does any response on which the verifier raises an exception. Responses truncated at the generation limit therefore score 0 and are kept as training examples. Only rollouts cancelled by the system are excluded from gradient steps. The system prompt states the remaining token budget.

Benchmark corrections and improvements. We corrected defects in the original data and verifiers that affected training rewards or evaluation scores.

- We removed 13,664 duplicate training instances, as well as 498 training and 1 validation function-induction instances for which the provided examples were insufficient to determine a unique answer.
- We removed the crossword family, whose puzzles were machine-assembled from a pool of clues with fabricated filler letters. Counting an entry as a word if its `wordfreq` Zipf frequency is at least 1.0, only 2.8% of the training crosswords had solutions in which every entry was a word. In addition, 84.8% of the clue–answer pairs in the evaluation splits appeared verbatim in training.
- The *fifteen-puzzle* generator's solvability test contained two sign errors, so it admitted 3,070 unsolvable boards among 6,024. Always answering "no solution" therefore earned about half of the family's score. We replaced these boards with boards generated by walking back from the solved state. For depths up to 14, an exhaustive breadth-first search certifies the optimal solution length. For deeper boards, iterative deepening A* verifies the target distance.
- We rewrote prompts and regenerated instances for 5 families whose stated rules did not match their verifiers. For *countdown*, we removed a rule that intermediate results be positive integers, which 27.2% of reference solutions violate. For *light up*, we added the missing requirement that every empty cell be lit. For *magic square*, we stated that all entries are distinct. For *minesweeper*, we asked for all determinable cells. For *slant*, we corrected the diagonal-symbol descriptions.
- We fixed verifier defects in 13 families. For *magic square*, the verifier rejected negative entries, so it rejected 48.8% of its own reference answers. For the two *Hamiltonian path* families, a "no path" answer could never be accepted. For *light up* and *natural-language navigation*, the verifier required an exact match to one stored solution although other solutions are valid. For *car painting*, the reference answer was a suboptimal heuristic solution that can be beaten. For the four *sliding-puzzle* families, solvability was never derived from the board. For *minesweeper*, the verifier compared answers against an incomplete reference and often rejected correct solutions that identified more mines. For *countdown* and *game24*, the verifier accepted operators outside $+, -, \times, \div$ contradicting the task definition and enabling reward hacking. The fixed verifiers check the puzzle constraints directly. We remove the reasoning trace before verification and assign a score of 0 to responses that exhaust the generation budget before completing the trace.

All other task instances not mentioned above are unchanged. Figure 6 compares independent RLVR learners trained on the original and corrected Enigmata data. To make the original-data results more comparable to the corrected split, we recompute their aggregate accuracy from the recorded per-family accuracies, weighting each family by its number of tasks in the corrected split and excluding crosswords (since bug fixes and filtering slightly changed the task distribution). This only changes

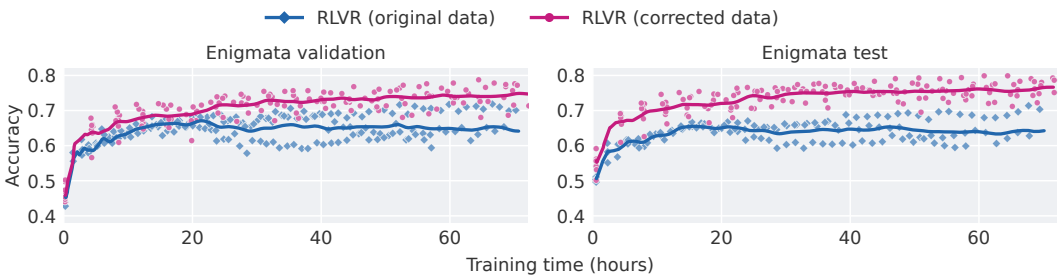


Figure 6: Enigmata results before and after the benchmark corrections of Section A.1.1. We compare independent RLVR learners trained on the original and corrected data. Original-data scores are reweighted to match the corrected splits' family composition, without re-evaluating checkpoints. Scatter points show subsampled evaluation results, and solid lines show smoothed means of the latest available score from each learner.

how the existing family-level results are aggregated; we do not re-evaluate the original checkpoints on the corrected tasks or verifiers. The comparison should therefore be viewed as approximate.

A.1.2 NEMOTRON

Data. Nemotron (NVIDIA, 2025) is NVIDIA's Nemotron-3-Nano RL training blend of 93,244 instances across 7 task subsets. We use the 6 single-turn subsets: STEM multiple-choice questions, competition mathematics from DAPO-Math-17k and Skywork-OR1, instruction following, JSON generation against a supplied schema, and competitive programming. The programming tasks contain up to 50 tests each. We exclude the multi-turn workplace-assistant subset, as multi-turn tool use is covered by Gaia2 and AppWorld. The released blend omits the mathematics prompts, so we restore the missing 22,056 prompts from the source datasets. We construct training, validation, and test sets with no shared prompts. Validation tasks are selected from instances whose prompts do not occur in the training pool. Test tasks drawn from the training pool are removed together with all copies of their prompts. The splits contain 81,725 training tasks and 512 tasks each for validation and test. Validation and test contain the same number of tasks from each subset. Each evaluation split contains 160 multiple-choice, 139 instruction-following, 97 Skywork, 43 JSON, 39 programming, and 34 DAPO tasks.

Evaluation. We use task-specific rule-based verifiers without an LLM judge. Multiple-choice answers must match the answer patterns supplied with each task, and mathematical answers are verified symbolically. Instruction-following responses must satisfy all listed constraints, JSON responses must validate against their supplied schemas, and programming responses must pass all tests. We score only the text after the final `</think>` tag, which marks the end of the reasoning trace. A response with an unclosed reasoning trace scores 0. We sample 2 rollouts per task on both the validation and test splits and report mean accuracy over all rollouts. A correct response scores 1, and an incorrect response scores 0. Responses that exhaust the generation budget without an answer or cannot be graded by the verifier also score 0 rather than being excluded. The initial evaluation temperature is 0.6, with top-p set to 0.95 and top-k to 20. We do not use multi-token prediction. The context and generation limits are fixed at 48,000 and 32,768 tokens, respectively. Validation and test use the member's current evaluation temperature. After a PBPT action, the evaluation process restarts with the new configuration and evaluates only checkpoints trained after that action. Only validation accuracy is used as fitness for population decisions.

We use $p_{\text{val}} = 2$ and $p_{\text{test}} = 6$ under the evaluation schedule above. Selection resets the pass counter, so the schedule need not remain periodic across population decisions. Within the plotted interval, successive validation evaluations are a median of 3 training iterations apart for PBPT and 4 for independent RLVR, compared with 11 and 15 for test.

Reward. We use the same task-specific verifier to assign a binary training reward: 1 for an accepted response and 0 otherwise. Programming tasks receive no partial credit for passing only some tests. If the verifier cannot grade a response, we assign 0 and retain the rollout in its group. There

are no additional length or format penalties. Responses truncated at the generation limit are graded and retained, while cancelled rollouts are excluded from policy updates. The system prompt states the generation budget available for the task.

A.1.3 GAIA2

Data. Gaia2 (Froger et al., 2026) uses the Mobile environment of Meta's ARE simulator, with email, chat, calendar, and contacts. Agents complete user requests by interacting with its 101 APIs. We use the public validation split, which contains 800 scenarios across the 5 capability categories listed in Section 5. We split these scenarios into training, validation, and test sets by universe, keeping the underlying application data disjoint. Universes 21, 23, 26, 27, 28, and 30 provide 458 training scenarios, universes 24 and 29 provide 153 validation scenarios, and universes 22 and 25 provide 189 test scenarios. All splits use the same applications and tools.

Evaluation. We use Qwen3.5-9B for all LLM judges. The judge is the base model without the policy's LoRA adapter, served on the learner's own inference servers. The judges use thinking mode and temperature 0.5. Each judge call generates one response of up to 16,384 tokens. We sample 2 rollouts per validation scenario and 1 per test scenario. We report task goal completion, the fraction of successful rollouts. The initial policy evaluation temperature is 0.5, without top-p or top-k filtering. Each policy call may initially generate up to 6,000 tokens, and PBPT mutates this limit within $[4{,}000, 10{,}000]$. We do not use multi-token prediction. Each rollout has a maximum of 200 steps, matching the original benchmark configuration. We use $p_{\text{val}} = 1$ and $p_{\text{test}} = 4$, so every pass includes validation. Because Gaia2 evaluation is slow, consecutive validated checkpoints are a median of 3 training iterations (about 1.1 hours) apart, and test evaluations about 11 iterations (about 5 hours) apart, for both methods.

Judging procedure. We use ARE's per-event judge to compare the agent's actions against the task's reference (oracle) events. For each turn, the number of calls to each tool must match the oracle's, except that the agent may send one additional message to the user. Each oracle event is then matched to a distinct agent event. A match requires the agent event to fall within a timing window relative to the events it depends on and to pass deterministic checks on its arguments. It must also pass LLM-based checks of properties such as content, plausibility, and tone. Each oracle event on which it depends must also be matched to an earlier agent event. A rollout succeeds if and only if every oracle event of every turn is matched. Each LLM-based check receives the agent's and the oracle's action arguments with task context and returns a Boolean verdict with a rationale. If no verdict parses within 3 attempts, the rollout is scored as a failure and retained for training.

Policy-side timeouts are handled the same way: a generation stall of 300 seconds, or a total generation time or rollout wall-clock time of 1,800 seconds ends the rollout as a failure. Tool errors returned by the simulated applications are ordinary observations. Only rollouts affected by infrastructure failures are discarded without a reward. These are judge connection errors or timeouts and rollout processes killed after 2,400 seconds.

Reward. The training reward is binary: 1 for success and 0 for failure. Training and evaluation use the same judging procedure and settings. They differ only in the policy's sampling temperature, the number of rollouts per scenario, and the number of concurrent scenario runners.

Versions and modifications. We use a fork of ARE and the public Gaia2 scenarios released on Hugging Face as `meta-agents-research-environments/gaia2`. Relative to upstream, our fork makes the following changes:

- LLM judges return schema-validated tool-call verdicts with 1 vote instead of 3 due to compute cost considerations.
- We recover verdicts from judge text and strip reasoning traces before subtask extraction.
- LLM-based checks run in parallel where possible for performance reasons.
- The agent uses native tool calling rather than a text ReAct loop.
- The agent's system prompt clarifies reporting to the user and waiting for notifications.

- Observations are truncated, and wall-clock caps are added.
- Judge infrastructure failures are classified separately from judge verdicts.
- The step limit is applied as configured.

A.1.4 APPWORLD

Data. AppWorld (Trivedi et al., 2024) evaluates personal-assistant agents in a simulator with 10 applications, including Amazon, Spotify, Venmo, and Gmail. Each task consists of a user request, such as transferring money or gathering listening statistics. The agent fulfills the request through successive Python code executions that call application APIs. We use version `0.1.3` of the `appworld` package. We use the `train` split with 90 tasks for training. For validation, we use 55 of the 57 tasks in the `dev` split, excluding tasks `3ab5b8b_3` and `37a8675_1` because of bugs in their task definitions. We report results separately on `test_normal` with 168 tasks and `test_challenge` with 417 tasks.

Evaluation. We sample 2 rollouts per validation task and 1 per task in each test split. A task succeeds only if all required checks on the response and final application state pass. We report task goal completion, the fraction of successful rollouts. The initial evaluation temperature is 0.1, without top-*p* or top-*k* filtering, with a context limit of 65,536 tokens and a generation limit of 8,192 tokens per policy call. We allow up to 30 interactions with the Python interpreter during training and 50 during evaluation. We use $p_{\text{val}} = 1$ and $p_{\text{test}} = 3$, so every pass includes validation. Consecutive validated checkpoints are a median of 1 training iteration (0.45 hours for PBPT and 0.6 hours for independent RLVR) apart, and test evaluations 3 to 4 iterations (about 1.5 to 1.8 hours) apart. There is no wall-clock limit on the policy. An error in the agent's own code is returned to the agent as an observation. Infrastructure errors, such as a failed or timed-out call to the AppWorld server (100 seconds per call), restart the server and rerun the episode from scratch, up to 5 attempts. If all attempts fail, the whole training iteration or evaluation pass is aborted rather than silently dropping the rollout.

Reward. At the end of each rollout, the training reward is the fraction of task-specific checks passed. For example, passing 3 of 4 checks earns 0.75 but does not count as success in evaluation. We also score rollouts that exhaust interaction or context budgets, with no extra penalty.

A.2 RL CONFIGURATIONS AND SEARCH SPACES

Within each comparison, all methods use the same configuration before initial mutation. The initial values in the tables below are launch values, identical for every member and seed. PBPT applies an initial mutation to 5 of its 8 members in the main experiments and to 6 of 8 in the suboptimal-configuration experiment. The initial mutation uses the same mutation probability and scaling range as later mutations. Independent RLVR and PBPT without mutation do not apply an initial mutation.

A.2.1 MAIN EXPERIMENTS

Shared settings. Tables 2 and 3 give the initial configurations and PBPT search ranges for the reasoning and agentic tool-use benchmarks, respectively. PBPT uses $p_{\text{mut}} = 0.2$ and numeric scaling factors $\mu \sim U[1.1, 1.5]$. Section A.3 describes the mutation operator.

Benchmark-specific settings. Enigmata disables scenario replay and includes 35 mutable task-family sampling weights and 2 mutable scenario-runner counts. On the other benchmarks, scenario replay is initially enabled, and task-family weights and scenario-runner counts are not mutated.

Fixed RL settings. All experiments use AdamW with $(\beta_1, \beta_2) = (0.9, 0.95)$ and $\epsilon = 10^{-15}$. The learning rate is constant between PBPT actions, without warm-up or decay. Training uses bfloat16 and LoRA adapters with rank 8, scaling factor 16, and no dropout on all attention, feed-forward, and linear-attention projections. Gradients are clipped to the configured maximum norm. Advantages use a leave-one-out baseline without normalization. There is no Kullback–Leibler (KL) penalty and no entropy bonus. Under GSPO, each iteration runs the configured number of epochs over minibatches, with sequence-level clipping to $[1 - \epsilon_{\text{low}}, 1 + \epsilon_{\text{high}}]$. Under RLOO, training is strictly

Table 2: Initial RL configurations and PBPT search ranges for the reasoning benchmarks. Values precede initial mutation, "fixed" means not mutated, and "—" denotes inapplicable settings.

	Enigmata (4B)		Nemotron (4B)	
Hyperparameter	Initial value	Search range	Initial value	Search range
Optimization				
Learning rate	5×10^{-5}	$[10^{-8}, 10^{-3}]$	5×10^{-5}	$[10^{-8}, 10^{-3}]$
Weight decay	0.01	$[10^{-5}, 0.02]$	0.01	$[10^{-5}, 0.02]$
RL core				
RL objective	GSPO	{RLOO, GSPO}	GSPO	{RLOO, GSPO}
Epochs per iteration	2	$[1, 4]$	2	$[1, 4]$
Rollouts per scenario	8	$[3, 16]$	8	$[3, 16]$
Scenarios per iteration	40	$[8, 96]$	40	$[8, 96]$
Minibatch size	64	$[8, 256]$	64	$[8, 256]$
Max gradient norm	0.1	$[10^{-6}, 1]$	0.1	$[10^{-6}, 1]$
Trust region				
Lower clipping parameter ϵ_{low}	3×10^{-4}	$[2 \times 10^{-4}, 0.01]$	3×10^{-4}	$[2 \times 10^{-4}, 0.01]$
Upper clipping parameter ϵ_{high}	4×10^{-4}	$[2 \times 10^{-4}, 0.01]$	4×10^{-4}	$[2 \times 10^{-4}, 0.01]$
Truncated importance sampling c	2	$[1, 10]$	2	$[1, 10]$
Clip log-prob grads at $-\epsilon$	on	{on, off}	on	{on, off}
Entropy				
REPO-R rescaling	off	{on, off}	off	{on, off}
REPO-R $\beta_{\max}$	0.05	$[0.01, 0.2]$	0.05	$[0.01, 0.2]$
Rollout sampling				
Non-zero-advantage fraction	0.25	$[0.1, 0.5]$	0.25	$[0.1, 0.5]$
Rollouts fraction	0.9	$[0.1, 0.99]$	0.9	$[0.1, 0.99]$
Rollouts-per-scenario fraction	0.66	$[0.1, 0.99]$	0.66	$[0.1, 0.99]$
Scenario replay				
Scenario replay	off	fixed	on	{on, off}
Replay fraction	—	—	0.3	$[0.05, 0.9]$
Replay max fail streak	—	—	5	$[2, 20]$
Evaluation				
Evaluation temperature	0.6	$[0.05, 0.75]$	0.6	$[0.05, 0.75]$
Concurrency				
Training scenario runners	80	$[20, 200]$	80	fixed
Evaluation scenario runners	200	$[50, 400]$	200	fixed
Generation budget				
Maximum training sequence length	48,000	fixed	48,000	fixed
Maximum generated tokens	24,576	fixed	32,768	fixed
Task curriculum				
Per-family sampling weight	1.0 (unset)	$[0.05, 20]$	—	—

on-policy with a single epoch, a single optimizer step, and no GSPO clipping. In that case, the stored epoch count, minibatch size, and clipping bounds have no effect. Truncated importance sampling, used under both objectives, corrects for numerical differences between the vLLM sampler and the trainer. Let p_ψ denote the token probabilities reported by the vLLM sampler, and let $p_{\theta_{\text{old}}}$ denote the probabilities recomputed by the trainer with its parameters at the start of the iteration. Each output token x_t's loss term is multiplied by the fixed weight $\min\{p_{\theta_{\text{old}}}(x_t \mid \mathbf{c}, \mathbf{x}_{<t})/p_\psi(x_t \mid \mathbf{c}, \mathbf{x}_{<t}),\, c\}$. GSPO computes its sequence-level importance ratio relative to $p_{\theta_{\text{old}}}$ rather than p_ψ. On Nemotron, training uses fully sharded data parallelism across 4 GPUs, and rollouts with absolute advantage below 0.01 are excluded from policy updates. Nemotron aborts iterations with absolute estimated KL divergence above 20. Nemotron uses no frequency penalty and a replay buffer of up to 20,000 scenarios.

Implementation-specific parameters. The *REPO-R* entropy regularizer (Petrenko et al., 2026) replaces each rollout's leave-one-out advantage A with a per-token advantage that depends on the detached token log-probability $\log p_t$. If $A > 0$, then $A_t = \max\{0, A(1 - \beta \log p_t)\}$, and if $A < 0$, then $A_t = \min\{0, A(1 + \beta \log p_t)\}$. Starting from $\beta = 10^{-3}$, β is doubled or halved once per iteration to steer the estimated token entropy toward its value at iteration 0. Its sign changes when $|\beta|$ falls below 10^{-4}, and $|\beta| \leq \beta_{\max}$. $\beta_{\max}$ has no effect while REPO-R is off.

Table 3: Initial RL configurations and PBPT search ranges for the agentic tool-use benchmarks. Values precede initial mutation, and "fixed" means not mutated.

	Gaia2 (9B)		AppWorld (4B)	
Hyperparameter	Initial value	Search range	Initial value	Search range
Optimization				
Learning rate	5×10^{-5}	$[10^{-8}, 10^{-3}]$	5×10^{-5}	$[10^{-8}, 10^{-3}]$
Weight decay	0.01	$[10^{-5}, 0.02]$	0.01	$[10^{-5}, 0.02]$
RL core				
RL objective	GSPO	{RLOO, GSPO}	GSPO	{RLOO, GSPO}
Epochs per iteration	2	$[1, 4]$	2	$[1, 4]$
Rollouts per scenario	6	$[3, 16]$	6	$[3, 16]$
Scenarios per iteration	32	$[8, 96]$	64	$[8, 96]$
Minibatch size	64	$[8, 256]$	128	$[8, 256]$
Max gradient norm	0.1	$[10^{-6}, 1]$	0.1	$[10^{-6}, 1]$
Trust region				
Lower clipping parameter ϵ_{low}	3×10^{-4}	$[2 \times 10^{-4}, 0.01]$	3×10^{-4}	$[2 \times 10^{-4}, 0.01]$
Upper clipping parameter ϵ_{high}	4×10^{-4}	$[2 \times 10^{-4}, 0.01]$	4×10^{-4}	$[2 \times 10^{-4}, 0.01]$
Truncated importance sampling c	2	$[1, 10]$	2	$[1, 10]$
Clip log-prob grads at $-\epsilon$	on	{on, off}	on	{on, off}
Entropy				
REPO-R rescaling	off	{on, off}	off	{on, off}
REPO-R $\beta_{\max}$	0.05	$[0.01, 0.2]$	0.05	$[0.01, 0.2]$
Rollout sampling				
Non-zero-advantage fraction	0.25	$[0.1, 0.5]$	0.25	$[0.1, 0.5]$
Rollouts fraction	0.9	$[0.1, 0.99]$	0.9	$[0.1, 0.99]$
Rollouts-per-scenario fraction	0.66	$[0.1, 0.99]$	0.66	$[0.1, 0.99]$
Scenario replay				
Scenario replay	on	{on, off}	on	{on, off}
Replay fraction	0.3	$[0.05, 0.9]$	0.3	$[0.05, 0.9]$
Replay max fail streak	5	$[2, 20]$	5	$[2, 20]$
Evaluation				
Evaluation temperature	0.5	$[0.1, 0.75]$	0.1	$[0.05, 0.75]$
Concurrency				
Training scenario runners	16	fixed	24	fixed
Evaluation scenario runners	36	fixed	64	fixed
Generation budget				
Maximum training sequence length	80,000	fixed	65,536	[48,000, 65,536]
Maximum generated tokens	6,000	[4,000, 10,000]	8,192	[4,000, 9,000]

Log-probability gradient clipping stops gradients through token log-probabilities of at least $-\epsilon$, where ϵ is the float32 machine epsilon. This works around numerical issues for near-certain tokens.

The *rollout collection* target, rollouts, equals scenarios per iteration times rollouts per scenario. A scenario's group is eligible for collection once at least $\max\{2, \text{round}(f_{\text{rps}} \cdot \text{rollouts per scenario})\}$ of its rollouts complete, where f_{rps} is the rollouts-per-scenario fraction. Collection stops once eligible groups contain at least $f_{\text{r}} \cdot \text{rollouts}$ completed rollouts, where f_{r} is the rollouts fraction. At least $f_{\text{nz}} \cdot \text{rollouts}$ of these must come from groups with non-zero advantage, where f_{nz} is the non-zero-advantage fraction. Unfinished rollouts are then cancelled. Each completed group with zero advantages triggers sampling of a replacement scenario, up to $10 \cdot \text{rollouts}$ collected rollouts. Zero-advantage rollouts are excluded from the update.

Scenario replay stores task instances whose last group contains both successful and failed rollouts. Each scenario draw comes from the buffer with probability equal to the replay fraction. Instances leave after a fully successful group or the maximum failure streak, measured in consecutive fully failed groups. On AppWorld, a rollout counts as solved only if all checks pass.

The *scenario-runner* counts are the numbers of concurrent rollout workers. For training, the count is per learner process, with 4 processes per learner. For evaluation, it is the total count. All workers share the learner's vLLM servers.

Table 4: Initial RL configuration and PBPT search ranges for the suboptimal-configuration experiments on Enigmata in Sections 5.2 and 5.3. The context limit (48,000 tokens) and generation limit (24,576 tokens) are fixed. "Final" summarizes the PBPT population at the end of training. For numeric hyperparameters, it gives the median over the values recorded during each member's last 100 training iterations, pooled across the 8 members. For Boolean settings, it gives the number of members ending with the setting on. "Mixed" indicates that members ended with different RL objectives (6 RLOO, 2 GSPO). Under RLOO, the epoch count, minibatch size, and clipping bounds are inactive. The per-family weights give the median and the 10th–90th percentiles over the final weights of all 35 families and 8 members.

Hyperparameter	Initial value	Search range	Final
Optimization			
Learning rate	10^{-4}	$[10^{-8}, 10^{-3}]$	2.7×10^{-5}
Weight decay	2.5×10^{-3}	$[10^{-5}, 0.02]$	3.2×10^{-3}
RL core			
RL objective	GSPO	{RLOO, GSPO}	mixed
Epochs per iteration	4	$[1, 4]$	3
Rollouts per scenario	4	$[3, 16]$	10
Scenarios per iteration	48	$[8, 96]$	40
Minibatch size	24	$[8, 256]$	80
Max gradient norm	0.5	$[10^{-6}, 1]$	0.223
Trust region			
Lower clipping parameter ϵ_{low}	2×10^{-3}	$[2 \times 10^{-4}, 0.01]$	1.5×10^{-3}
Upper clipping parameter ϵ_{high}	2×10^{-3}	$[2 \times 10^{-4}, 0.01]$	4.2×10^{-3}
Truncated importance sampling c	6	$[1, 10]$	1.28
Clip log-prob grads at $-\epsilon$	on	{on, off}	2/8 on
Entropy			
REPO-R rescaling	off	{on, off}	4/8 on
REPO-R $\beta_{\max}$	0.05	$[0.01, 0.2]$	0.196
Rollout sampling			
Non-zero-advantage fraction	0.12	$[0.1, 0.5]$	0.149
Rollouts fraction	0.98	$[0.1, 0.99]$	0.985
Rollouts-per-scenario fraction	0.85	$[0.1, 0.99]$	0.26
Evaluation			
Evaluation temperature	0.1	$[0.05, 0.75]$	0.0574
Concurrency			
Training scenario runners	30	$[20, 200]$	30
Evaluation scenario runners	200	$[50, 400]$	184
Task curriculum			
Per-family sampling weight (35 families)	1.0 (unset)	$[0.05, 20]$	1.26 $[0.14, 5.28]$

Enigmata uses *task-family sampling weights* w_f to draw family f with probability $w_f / \sum_g w_g$, then samples an instance uniformly within that family. Setting all weights to 1 samples families uniformly, not individual instances.

A.2.2 Adaptation from Suboptimal RL Configurations

Table 4 gives the suboptimal initial configuration used in Sections 5.2 and 5.3. We choose these values manually within PBPT's search ranges to represent plausible settings that can be chosen prior to careful hyperparameter tuning. All methods start from this configuration, with initial mutation applied only to PBPT as described above. For these experiments, PBPT uses $p_{\text{mut}} = 0.3$ and numeric scaling factors $\mu \sim U[1.1, 2.5]$. PBPT without mutation uses the same selection rule but no mutation during training. All other population-decision settings match the main experiments (Section A.3).

A.2.3 Scaling PBPT

This section studies how PBPT behaves with a larger population of 16 members and analyzes how the population's hyperparameters and training behavior evolve over the course of a run. Table 5 com-

pares 16 members with the 8-member run from Table 1. The two runs differ only in population size. The larger population reaches higher accuracy and explores roughly $1.8\times$ as many configurations.

Table 5: PBPT with 8 and 16 members on Enigmata. Best val. and test are the highest single evaluations (%) of any member within 66 hours. Counts cover the full runs, with events summed over members. Configurations are deduplicated across members, excluding task-mixture weights and evaluation-only settings. Selection events correspond to SELECT-AND-MUTATE actions.

Members	Best val.	Δ	Best test	Δ	Distinct configs	Mutation events	Selection events
8	80.35	–	79.36	–	205	204	82
16	**82.10**	+1.75	**80.96**	+1.60	363	388	180

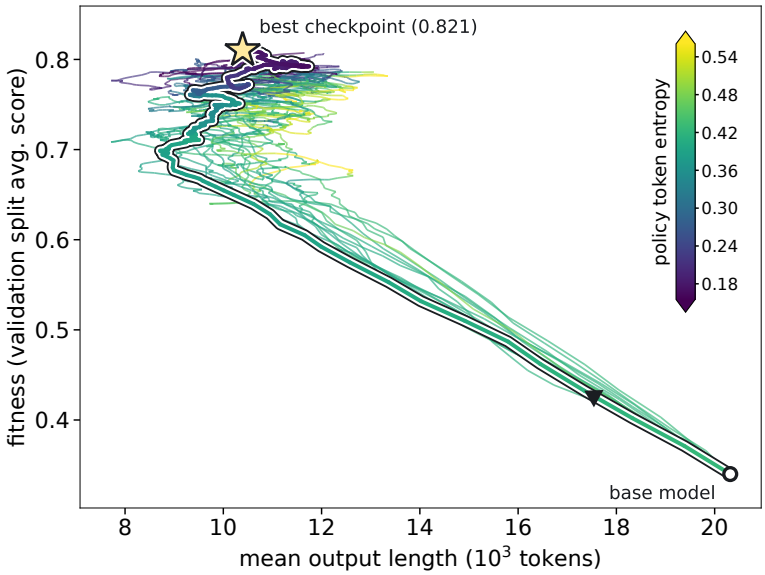


Figure 7: Trajectories of the 16-member population on Enigmata, colored by policy token entropy. The outlined line is the ancestry of the best checkpoint, followed through selection events.

A.2.4 PBPT BEHAVIOR VISUALIZATION

Figure 7 shows the population's training behavior. Output length drops quickly from the base model and plateaus, after which accuracy gains track falling policy entropy (Spearman $\rho = -0.71$).

Figure 8 shows the configurations explored by the 16-member population, analogously to Figure 5c. The visualization is inspired by low-dimensional embedding methods such as t-SNE, but uses manually constructed axes to preserve interpretability. Each configuration explored by PBPT is projected onto two directions in hyperparameter space defined by the weights in Table 6. Each numeric hyperparameter is $\log_{10}$-transformed where strictly positive and standardized over all configurations in the run. Each axis is then computed as a weighted mean of the standardized values, normalized by the sum of absolute weights. Positive weights move a configuration toward either larger optimization steps or more precise, computationally expensive gradient estimates. Weight decay and evaluation settings receive zero weight and are omitted because they do not directly affect either axis.

A.3 PBPT IMPLEMENTATION DETAILS

Unless stated otherwise, these settings apply to all benchmarks and experiments. Algorithm 2 gives the decision, crossover, and mutation operators used in Algorithm 1. For single population decisions in this subsection, we omit the evaluation index k. We write $\mathrm{fitness}\,(r)$ and $\mathrm{time}\,(r)$ for record r's validation score and runner process time. Std denotes the standard deviation of scores in $\mathcal{C}_i$.

Hyperparameter	Step size	Gradient
Learning rate	$+1.00$	—
Epochs per iteration	$+0.60$	—
Rollouts per scenario	—	$+0.80$
Scenarios per iteration	—	$+1.00$
Parallel scenario runners	$+0.15$	—
Minibatch size	-0.60	$+0.60$
Max gradient norm	$+0.70$	—
Lower clipping parameter ϵ_{low}	$+0.35$	—
Upper clipping parameter ϵ_{high}	$+0.35$	—
Truncated importance sampling c	$+0.30$	-0.40
Clip log-prob grads at $-\epsilon$	-0.30	—
REPO-R rescaling	-0.15	—
REPO-R $\beta_{\max}$	-0.15	—
Non-zero-advantage fraction	—	$+0.50$
Rollouts fraction	—	$+0.25$
Rollouts-per-scenario fraction	—	$+0.25$

Table 6: Axis weights for the hyperparameter maps in Figures 5c and 8.

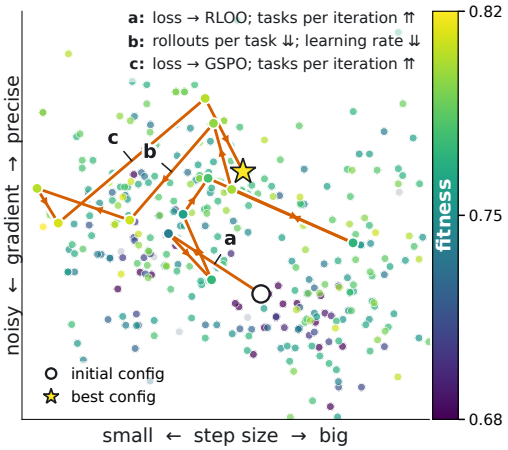


Figure 8: Configurations explored by the 16-member population, colored by best fitness. The line traces the ancestry of the best configuration, with letters marking its notable mutations.

Clocks. Each member has two clocks. The *runner process time* (time in Section 4) includes training, evaluation, and restarts but pauses while the job is suspended. It determines compute eligibility, the warm-up and cooldown intervals, and the per-member budget in Algorithm 1. The *training time* accumulates only the wall-clock time of training iterations (rollout collection and policy updates) and is the x-axis of the experimental figures. Selection preserves the recipient's clocks and iteration counter. Iteration-based checks use the evaluated checkpoint's training-iteration index.

Peer comparisons. A new validation evaluation triggers a population decision for member i. For each peer, member i takes the record with the greatest process time not exceeding time_i, as described in Section 4.2. Each record also stores a wall-clock publication timestamp for recency checks. The peer is excluded only if this record's process time lags time_i by more than 2 hours and it was published more than 2 hours ago in wall-clock time.

Decision eligibility. For a population of n members, at least $\lfloor n/2 \rfloor$ members, including member i itself, must be represented in $\mathcal{C}_i$ before selection or mutation is allowed. Selection and mutation require at least 30 minutes of runner process time and an evaluated checkpoint from training iteration 3 or later. After SELECT-AND-MUTATE or a mutation that changes the configuration, further selection or mutation requires at least 40 more minutes of runner process time and 2 more training iterations (1 on Gaia2). Initial mutation also starts this cooldown, so initially mutated members become eligible no earlier than 40 minutes, compared with 30 minutes for the others. Mutations that leave the configuration unchanged do not restart the cooldown.

Population decisions. With all 8 members represented, the winner and loser sets in Algorithm 2 contain 3 and 2 members, respectively. The *loser streak* counts consecutive calls to DECIDE that pass the eligibility checks above and place the member in $\mathcal{L}$, including the current call. Each eligible call uses its own comparison set. Calls that fail the checks neither extend nor break the streak, while an eligible call outside $\mathcal{L}$ resets it. The minimum streak is 1 in all our experiments.

Donor selection and state transfer. Every published record remains in $\mathcal{S}$ for the whole run. For selection, the donor pool combines two sources. The first is the records in $\mathcal{C}_i$ of the members in $\mathcal{W}$, all of which have higher fitness than the recipient because DECIDE otherwise returns KEEP. The second is up to 10 other records in $\mathcal{S}$ from any member, including the recipient. These records must have fitness at least best and process time not exceeding time_i, and they are chosen in descending order of fitness from records not already in the first group. The donor is drawn uniformly from the combined pool. The recipient copies the donor's policy parameters and optimizer state, applies CROSSOVER to the two configurations, and mutates the result as in Algorithm 1.

Algorithm 2. PBPT Decision, Crossover, and Mutation Operators

function DECIDE($i, \mathcal{C}_i, \mathcal{S}$)
  if local progress or peer evidence is insufficient **then**
    return (KEEP, $\varnothing$)
  Let $(\text{fitness}_i, \text{time}_i) \leftarrow$ the score and time in member i's record in $\mathcal{C}_i$
  Let $\mathcal{W}$ contain the highest-fitness $\max\{1, \lceil 0.3|\mathcal{C}_i| \rceil\}$ members represented in $\mathcal{C}_i$
  Let $\mathcal{L}$ contain the lowest-fitness $\max\{1, \lceil 0.25|\mathcal{C}_i| \rceil\}$ members represented in $\mathcal{C}_i$
  if $i \notin \mathcal{L}$ **then**
    return (KEEP, $\varnothing$)
  Let $\text{streak} \leftarrow$ the number of consecutive eligible calls with $i \in \mathcal{L}$
  if streak is below its minimum or $\text{fitness}_i \geq \min_{j \in \mathcal{W}} \text{fitness}_j$ **then**
    return (KEEP, $\varnothing$)
  Let $\text{best} \leftarrow \max_{r \in \mathcal{C}_i} \text{fitness}(r)$ and $\text{gap} \leftarrow \text{best} - \text{fitness}_i$
  Let $\text{spread} \leftarrow \text{Std}_{r \in \mathcal{C}_i}(\text{fitness}(r))$
  Let $\tau_{\text{sel}} \leftarrow \max\{\text{spread}, 0.05\,|\text{best}|\}$
  Let $\tau_{\text{mut}} \leftarrow \max\{0.1\text{spread}, 0.02\,|\text{best}|\}$
  if $\text{gap} > \tau_{\text{sel}}$ **then**
    Let $\mathcal{D}$ contain the records in $\mathcal{C}_i$ of the members in $\mathcal{W}$
    Add to $\mathcal{D}$ up to 10 highest-fitness records from $\mathcal{S} \setminus \mathcal{D}$, restricted to records r satisfying $\text{fitness}(r) \geq \text{best}$ and $\text{time}(r) \leq \text{time}_i$
    Sample donor uniformly from $\mathcal{D}$
    return (SELECT-AND-MUTATE, donor)
  if $\text{gap} > \tau_{\text{mut}}$ **then**
    return (MUTATE, $\varnothing$)
  return (KEEP, $\varnothing$)

function CROSSOVER(h, h_{donor})
  for each coordinate q of h **do**
    Let $h_{\text{new},q} \leftarrow h_q$ or $h_{\text{donor},q}$, each with probability $1/2$, independently
  return h_{new}

function MUTATE(state)
  Let $(\theta, \omega, h) \leftarrow$ state
  for each mutable coordinate q of h **do**
    if a Bernoulli (p_{mut}) draw succeeds **then**
      Mutate h_q according to its declared type and valid range
  Enforce the cross-parameter constraints on h
  return (θ, ω, h)

After a training or evaluation crash, or after no new evaluation arrives for 2.5 hours (4 hours on Gaia2), the member is forced to take SELECT-AND-MUTATE or MUTATE, outside the rule above.

Configuration mutation. Section A.2 specifies each experiment's mutation probability and numeric scaling bounds $\mu_{\min}$ and $\mu_{\max}$. Each mutable hyperparameter is selected independently with probability p_{mut}. Mutation flips Booleans and switches between GSPO and RLOO. A selected numeric value x is multiplied or divided, with equal probability, by $\mu \sim U[\mu_{\min}, \mu_{\max}]$. For fraction-valued parameters, the operation is applied to $\delta = \min(x, 1 - x)$, giving δ', and the new value is $1 - \delta'$ if $x > 0.5$ and δ' otherwise. The result is clamped to the search range. If clamping leaves the value unchanged because it already sat at the bound, the opposite operation is applied directly to the original value x, also for fraction-valued parameters, and clamped again.

Integer values are rounded and then rounded up to the parameter's multiple. The multiple is the number of learning GPUs (4) for scenarios per iteration and minibatch size, 10 for the Gaia2 and AppWorld generation limits, and 1 otherwise. If rounding returns the original value, one step of the

multiple is added or subtracted in the sampled direction before clamping. Thus, a selected integer always changes unless its range is a single value.

An unset per-family weight takes the sampler's default of 1.0. It is recorded only when first selected, and it is then mutated from 1.0 in the same draw. The only cross-parameter constraint raises ϵ_{high} to ϵ_{low} when a mutation would invert them.