Bootstrapping Language and Numerical Feedback for Reinforcement Learning in LLMs

Ang Li [*†1] **Yifei Wang** [*2] **Zhihang Yuan** [*3] **Stefanie Jegelka** [4] **Yisen Wang** [1]

Abstract

Reinforcement learning for large language models (LLMs) often relies on scalar rewards, a practice that discards valuable textual rationale buried in the rollouts and hampers training efficiency. While keeping and re-using the textual information seems intuitive to improve sample efficiency, we show that naive attempts are often counterproductive, risking either memorization from leaked solutions or behavior collapse due to irrelevant context. To address this, we propose *Language-And-Numerical Policy Optimization (LANPO)*, a framework that explicitly separates the roles of feedback: language guides exploration, while numerical rewards drive optimization. Central to LANPO is a dynamically evolving experience pool accumulated from on-policy rollouts, which facilitates two principles to ensure the effectiveness of feedback: *Reward-Agnostic Reflection* for safe intra-sample self-correction and *Relevant Abstraction* to distill generalizable lessons from inter-sample experiences. Across the evaluated mathematical reasoning and coding benchmarks, LANPO consistently outperforms strong GRPO-trained baselines for 7B and 14B models in zero-shot accuracy, and further improves performance with feedback-augmented inference at test time.

1. Introduction

Reinforcement learning (RL) has become a central ingredient for improving the reasoning abilities of large language models (LLMs) (OpenAI, 2024; Guo et al., 2025). In the prevalent pipeline, a model's complex reasoning is assessed by a programmatic verifier or an LLM-as-judge (Zheng et al., 2023; Gu et al., 2024), which compresses its evaluation into a single scalar reward. Policy optimization algorithms like PPO or its variants then update the model's parameters to maximize this scalar signal (Schulman et al., 2017a; Shao et al., 2024b). While effective, this scalarization of feedback discards the rich, explanatory rationale hidden in the model's textual responses. Consequently, exploration proceeds largely *de novo* for each prompt; the model cannot explicitly reason about why a previous attempt failed and must generate new rollouts without reusing these lesson-like experiences. This leads to repetitive, low-diversity exploration where failure patterns persist, causing state-of-the-art reasoning models to require thousands of RL steps to train (He et al., 2025).

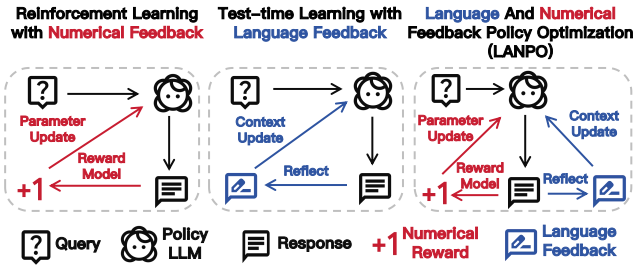


Figure 1. **Three paradigms for improving LLM reasoning. Left:** Standard RL uses scalar rewards to update parameters, but provides limited explicit guidance about why a rollout fails. **Middle:** Language feedback at inference time enables in-context adaptation (e.g., self-correction or in-context examples) without parameter updates. **Right:** *LANPO* combines both: language feedback conditions exploration by augmenting the rollout context, while scalar rewards drive policy optimization via parameter updates.

Unlike conventional RL agents, LLMs possess the unique ability to receive and process nuanced language feedback (Brown et al., 2020). This opens the door to learning from retrieved relevant knowledge or reasoning templates (Lewis et al., 2021; Yang et al., 2024b). However, naively integrating language feedback from historical rollouts into the RL training loop introduces critical challenges. On one hand, providing feedback from historical trials on the *same* problem (intra-sample feedback) risks *information leakage*; the model may learn to simply copy the correct answer, inflating training performance while undermining generalization. On the other hand, using feedback from *different* problems

[1]Peking University [2]MIT [3]ByteDance Seed [4]TUM. Correspondence to: Yisen Wang <yisen.wang@pku.edu.cn>.

Proceedings of the 43rd International Conference on Machine Learning, RLxF: Reinforcement Learning from World Feedback Workshop, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

(inter-sample feedback) often leads to *behavior collapse*, where the policy often under-utilizes retrieved context, generating near-identical outputs with/without context. These dilemmas have left language feedback as an underutilized resource in mainstream LLM training.

To resolve this tension, we propose *Language-And-Numerical Policy Optimization (LANPO)*, a training paradigm that synergistically *bootstraps language and numerical feedback* to enhance learning efficiency. As depicted in Figure 1 (right), LANPO unifies these two signals: language feedback is used to guide and enrich exploration via context updates, while numerical rewards are retained to drive robust policy optimization through parameter updates. At its core, LANPO introduces an *experience pool* that accumulates and distills past trials into concise, reusable natural-language summaries. To prevent the pitfalls of naive integration, we introduce two key mechanisms: (1) **Reward-Agnostic Reflection** for intra-sample feedback, where the model critiques and refines its own past attempts without access to the ground truth, thereby preventing leakage. (2) **Relevant Abstraction** for inter-sample feedback, which filters for semantically similar problems and summarizes their solutions into high-level principles, ensuring the guidance is both useful and generalizable, thus avoiding behavior collapse.

In summary, our main contributions are threefold: **(1) Identification and Mitigation of Core Failure Modes: (Section 3)** We identify and empirically characterize two practical failure modes—information leakage (intra-sample) and behavior collapse (inter-sample)—that hinder naive reuse of rollout text in LLM RL. **(2) A Robust Implementation Framework: (Section 4)** We introduce LANPO, a framework that integrates language feedback into numerical RL via an online experience pool, together with two mechanisms addressing the challenges above: *Reward-Agnostic Reflection* and *Relevant Abstraction*. **(3) Empirical Validation of Effectiveness: (Section 5)** On mathematical reasoning and coding benchmarks, LANPO outperforms strong GRPO baselines for 7B and 14B models in zero-shot accuracy, and further benefits from feedback-augmented test-time inference, achieving absolute gains of up to 10 percentage points on AIME-24.

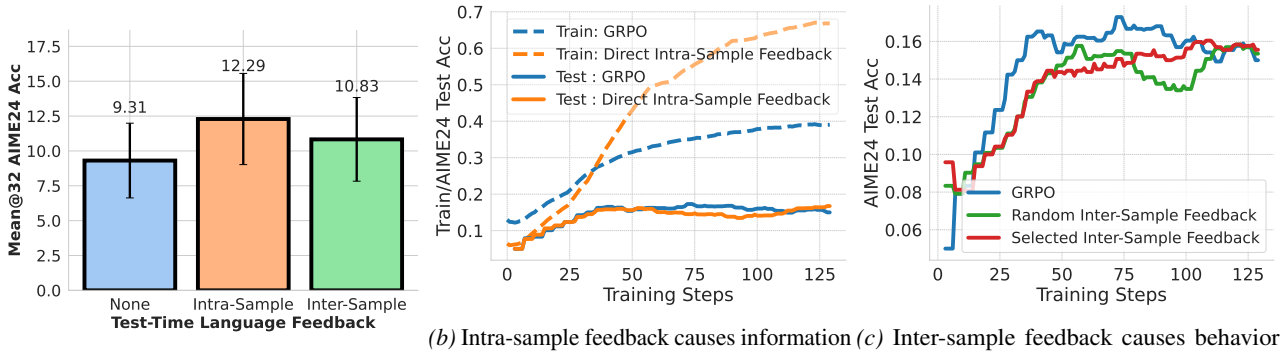
2. Related Work

Our work, LANPO, builds upon and intersects with three primary areas of research: Reinforcement Learning (RL) for LLMs, the use of language feedback for model improvement, and memory-augmented agent architectures.

RL with Numerical Feedback. The practice of optimizing LLMs with scalar rewards has become a cornerstone of developing advanced models (Ziegler et al., 2019), leading to powerful instruction-following chatbots (Ouyang et al., 2022; Bai et al., 2022) and specialized problem-solvers (OpenAI, 2024; Guo et al., 2025). The underlying algorithms have also evolved from Proximal Policy Optimization (PPO) (Schulman et al., 2017a) to more recent methods like Direct Preference Optimization (DPO) (Rafailov et al., 2023) and Group-Relative Policy Optimization (GRPO) (Shao et al., 2024b). Our work is orthogonal to the choice of the specific optimization algorithm. LANPO operates a level above, introducing a language-feedback layer that structures the context provided to the policy. This layer is designed to improve the quality and efficiency of exploration before the numerical reward is used for the policy update, making it a complementary component to any of these RL algorithms.

Learning with Language Feedback. Using language to refine model outputs is a well-explored area. This includes self-correction (Wang et al., 2024a; Kumar et al., 2024), generating self-critiques (Madaan et al., 2023; Yuan et al., 2024; Ankner et al., 2024), and maintaining reflections across episodes (Shinn et al., 2023; Yuan & Xie, 2025; Agrawal et al., 2025). Other works incorporate in-context examples as language feedback to guide generation during RL (Chen et al., 2024; Baronio et al., 2025; Li et al., 2025). LANPO's contribution lies in how it integrates language feedback into LLM RL as a self-contained training loop while overcoming specific failure modes. Our reward-agnostic reflection for intra-sample feedback differs from training models to self-correct (Kumar et al., 2024) by operating as a *single-turn* RL training. For inter-sample feedback, our relevant abstraction mechanism directly counteracts the behavior collapse that can occur when naively reusing raw solutions, providing feedback signals that co-evolve with the latest policy.

Memory-Augmented Language Agents. The adoption of an external memory to store and reuse past experiences is central to many advanced agents. These memories have been used to build skill libraries (Wang et al., 2023; Yang et al., 2025b), correct errors post-deployment (Madaan et al., 2022), and serve as an episodic "Case Bank" at test time (Wang et al., 2025). While LANPO's experience pool serves a similar function, it is uniquely designed for the RL training loop. Rather than just storing raw trajectories for retrieval, LANPO actively processes on-policy rollouts into abstracted summaries of principles and pitfalls. This distilled knowledge becomes a direct input for shaping exploration in subsequent RL episodes, creating a tight, synergistic loop between experience, exploration, and optimization that is absent in architectures where memory is primarily a deployment-time resource.



(a) Test-time gains from language feedback.

(b) Intra-sample feedback causes information leakage.

(c) Inter-sample feedback causes behavior collapse.

Figure 2. **Challenges in introducing language feedback to RL training. (a)** At test time, both intra-sample feedback (self-correction) and inter-sample feedback (in-context examples) yield accuracy gains for more than 1.5%. **(b)** In training, leakage-prone intra-sample feedback (e.g., containing gold answers) inflates training accuracy but does not improve test accuracy. **(c)** In training, reusing raw successful trajectories as inter-sample demonstrations often leads to behavior collapse and fails to outperform GRPO.

3. Challenges in Introducing Language Feedback

Large language models trained with RL typically generate thousands of rollouts per iteration, which are usually discarded after reward estimation. Yet, these rollouts contain rich intermediate reasoning steps and successful solution trajectories that could, in principle, serve as *language feedback* to guide exploration effectively. We hypothesize that if language feedback that improves test-time accuracy could be effectively integrated into training, it would accelerate policy search and improve performance on challenging training problems where vanilla RL struggles.

We therefore begin by revisiting the effectiveness of language feedback in inference. Using Qwen2.5-7B-Instruct (Yang et al., 2024a) on the AIME24 benchmark, we find that both *intra-sample feedback* (self-correction on wrong attempts (Madaan et al., 2023; Wang et al., 2024b)) and *inter-sample feedback* (in-context examples (Brown et al., 2020) retrieved from MATH500 (Hendrycks et al., 2021)) boost mean@32 accuracy compared to the baseline (Figure 2a).

This motivates us to explore incorporating these language feedback into RL training. However, our preliminary experiments reveal two critical obstacles when naively applying language feedback in training:

Intra-sample feedback risks information leakage. As a diagnostic control experiment, we incorporated the gold answers into the model's context as a simple form of language feedback, aiming to efficiently acquire correct reasoning trajectories. As shown in Figure 2b, training accuracy rises quickly, but test accuracy shows no improvement over GRPO. The model gradually learns to exploit the leaked labels rather than genuinely improve its reasoning ability. Moreover, at inference time, there is no oracle to indicate which solution to correct, making this strategy infeasible without a label-free design. In Appendix A.3, we repeat the study on intra-sample feedback with a less extreme variant of information leakage by revealing parts of ground truth solutions and observe similar trends.

Inter-sample feedback suffers from behavior collapse. We attempted to reuse correct trajectories discovered during training as in-context demonstrations for other problems, either by random sampling or by selecting similar problems. Both strategies, shown in Figure 2c, fail to outperform GRPO. Closer inspection reveals that the model often ignores the provided examples and directly outputs answers, a phenomenon we denote as *behavior collapse*. Because the reward depends only on final correctness, the policy can achieve similar returns by ignoring retrieved context; without explicit constraints, learning therefore gravitates toward a shorter, context-invariant solution path. This starkly contrasts with the effectiveness of in-context learning at inference, underscoring a disconnect between test-time and training-time dynamics. We present further discussion into behavior collapse in Appendix A.1.

In summary, while language feedback has clear potential to accelerate exploration in RL, naive integration during training introduces pitfalls: intra-sample feedback may risk exposing training shortcuts, while inter-sample feedback collapses into ineffective behavior. These challenges motivate the need for principled strategies to design language feedback mechanisms that can genuinely improve RL training—a direction we pursue in the next section.

4. LANPO: Language-And-Numerical Policy Optimization

Our preliminary study (Section 3) revealed that naïvely integrating language feedback into RL training leads to two failure modes: *information leakage* (when feedback contains shortcuts to rewards) and *behavior collapse* (when

reusing raw solutions as demonstrations). To overcome these pitfalls, we propose *Language-And-Numerical Policy Optimization (LANPO)*, a framework that establishes a synergistic loop between language and numerical feedback. LANPO introduces two core mechanisms—*Reward-Agnostic Reflection* and *Relevant Abstraction*—which transform language feedback into a safe and effective guide for exploration. Meanwhile, numerical rewards remain responsible for identifying and reinforcing high-quality behaviors through policy optimization. This separation of roles allows language to accelerate exploration by distilling knowledge from past trajectories, while numerical signals ensure stable learning, ultimately yielding a stronger policy that in turn generates better feedback for subsequent iterations.

4.1. Methodology

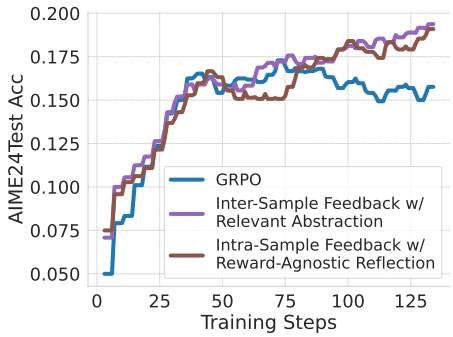


Figure 3. **Effectiveness of our algorithm design:** Inter-sample feedback benefits from relevance filtering and abstraction into transferable principles; intra-sample feedback uses label-free self-reflection over the model's own attempts to avoid leakage.

Reward-agnostic reflection for Intra-sample Feedback. Naïve intra-sample feedback, where the gold solution is revealed, inflates training accuracy but undermines generalization by encouraging memorization of leaked labels. LANPO replaces this with a *reward-agnostic reflection* mechanism. Instead of accessing the true label, the model revisits its own earlier attempts, critiques them step by step, and then produces a refined solution. This encourages reflective exploration while implemented within a single RL update step, without exposing gold labels.

Relevant Abstraction for Inter-sample Feedback. For inter-sample feedback, naïvely reusing raw solutions often triggers behavior collapse: the model learns to ignore the provided context and instead answers directly. LANPO overcomes this by introducing *relevant abstraction*, which ensures that reused experiences are both semantically aligned with the current problem and distilled into transferable knowledge. The process has three steps. First, *relevance filtering* restricts retrieval to trajectories drawn from sufficiently related problems, ensuring that the added context is useful. Second, *summarization and abstraction* condense raw solutions into high-level principles and common pitfalls that can generalize across problems, rather than problem-specific steps. Third, the actor is explicitly instructed to analyze the retrieved feedback before producing its own plan and solution, reinforcing active engagement with the context.

Together, reward-agnostic reflection and relevant abstraction transform the pitfalls of language feedback into guiding principles for exploration. As shown in Figure 3, both of the mechanisms make language feedback consistently beneficial compared to GRPO, avoiding collapse and yielding clear gains over naïve reuse of rollouts.

4.2. LANPO Modules and Training Objective

Building on the principles of *Reward-Agnostic Reflection* and *Relevant Abstraction*, LANPO organizes training around a modular pipeline. The design brings these ideas to life through a shared experience pool, two specialized responders, and a stable on-policy objective.

Experience pool. At the center of LANPO is a capped-size experience pool $\mathcal{E}_t$ that accumulates distilled experiences from past rollouts up to RL iteration t. Each entry contains a *structured summary* with a high-level solution outline, transferable principles and pitfalls, and metadata such as reward, source, and timestamp. When solving a new problem x, the policy retrieves context $c \sim p_c(\cdot \mid x, \mathcal{E}_t)$: existing attempts on the same x provide material for reflection, while filtered entries from semantically related problems supply abstracted guidance. In this way, the pool serves as a dynamic memory that fuels both reflection and abstraction. We provide complete implementation details of the experience pool in Appendix C.2.

Intra-sample exploration. LANPO introduces an intra-sample exploration module, which revisits the model's own earlier attempts, critiques them step by step, and refines the reasoning into a revised solution. This operationalizes Reward-Agnostic Reflection: the model learns to improve on its own outputs without ever knowing gold labels, fostering a self-reflective habit that strengthens exploration while avoiding leakage.

Inter-sample exploration. In parallel, to make retrieved context genuinely useful rather than ignored, LANPO equips the policy with an inter-sample exploration module. This component consumes summarized successful solutions from related problems and evaluates its relevance with the problem at hand. Only experiences with similarity-score higher than a predefined threshold γ will be incorporated into the policy's context. In doing so, it enforces the practice of drawing on transferable principles instead of copying raw solutions, directly realizing the idea of Relevant Abstraction. Summarization and filtering ensure that the retrieved

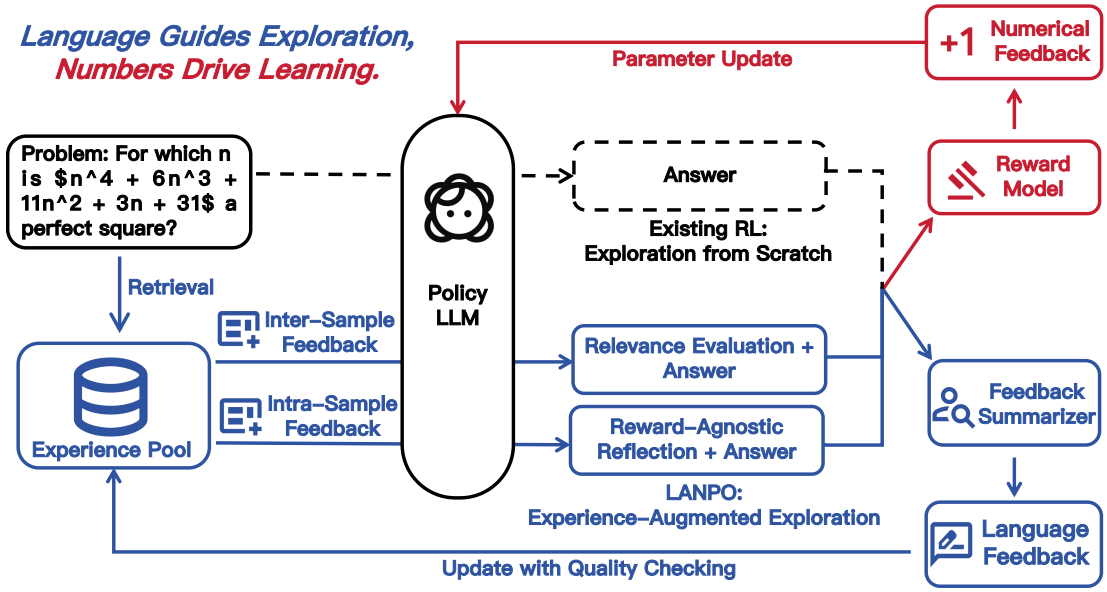


Figure 4. **LANPO training pipeline.** Language guides where to explore, while numerical rewards determine what to learn. **The language-guided exploration loop** shapes the policy's exploration. An experience pool stores principles and pitfalls from past trajectories. The actor uses this pool to perform inter-sample exploration (using guidance from related problems) and intra-sample reflection (critiquing its own past attempts). **The numerical-driven learning loop** optimizes the policy. All attempts, whether guided by language or not, are evaluated by a reward model. The numerical reward is then used to update the policy via policy gradient.

feedback is both relevant and generalizable, while the system prompt encourages active engagement with the context during reasoning.

Seeding atomic capabilities via SFT. Since neither summarization nor feedback-driven reasoning is innate to a base LLM, LANPO begins with a lightweight supervised fine-tuning (SFT) stage to provide the policy with three atomic skills: (i) a *Summarizer* that converts raw trajectories into concise entries for the pool, (ii) an *inter-sample exploration* capability that learns to evaluate and apply retrieved feedback, and (iii) an *intra-sample exploration* capability that learns to audit and refine past attempts. The purpose of this stage is not to maximize accuracy, but to instill literacy—the ability to write, read, and act on structured feedback—so that RL training can fully exploit the principles of reflection and abstraction.

RL Training loop and objective. During RL, the model alternates between feedback-aware rollouts and from-scratch rollouts. With probability p, a context c is drawn from $\mathcal{E}_t$; with probability $1-p$, c is the empty set $\emptyset$, which preserves the model's ability to reason without additional context.

Let τ_t denote the trajectories collected at iteration t (including both feedback-aware and from-scratch attempts). After collection, selected trajectories are summarized and synchronously added back into the pool, yielding an online-updated memory:

$$\mathcal{E}_{t+1} \leftarrow \mathcal{U}\big(\mathcal{E}_t, \mathcal{S}(\tau_t)\big), \tag{1}$$

where $\mathcal{S}$ is the abstraction operator and $\mathcal{U}$ implements pool maintenance (e.g., filtering, deduplication, and capacity control). As a result, the retrieval distribution $p_c(\cdot \mid x, \mathcal{E}_t)$ evolves over training.

The policy is updated using a GRPO-style surrogate objective (Schulman et al., 2017b; Shao et al., 2024a). Our key modification is to introduce an expectation over the context c, which is sampled from the language experience pool at the current iteration. This makes the standard GRPO objective feedback-aware while leaving the underlying surrogate unchanged. Concretely, the per-iteration objective is:

$$\mathcal{L}^{(t)}_{\text{LANPO}}(\theta) = \mathbb{E}_{x\sim\mathcal{D}}\, \mathbb{E}_{c\sim p_c(\cdot|x,\mathcal{E}_t)}\Big[\mathcal{L}^{\text{CLIP}}(\theta; x, c)\Big]. \tag{2}$$

where $\mathcal{L}^{\text{CLIP}}(\theta; x, c)$ is the context-dependent clipped surrogate objective from PPO, given a problem x sampled from the training dataset $\mathcal{D}$:

$$\mathcal{L}^{\text{CLIP}}(\theta; x, c) = \mathbb{E}_{y\sim\pi_{\theta_{\text{old}}}(\cdot|x,c)}\Big[\min\Big(\rho(\theta; x, c)\hat{A}, \\ \text{clip}\left(\rho(\theta; x, c), 1-\epsilon, 1+\epsilon\right)\hat{A}\Big)\Big]. \tag{3}$$

The probability ratio $\rho(\theta; x, c) = \frac{\pi_\theta(y|x,c)}{\pi_{\theta_{\text{old}}}(y|x,c)}$ compares the new policy to the old one used for data collection, conditioned on the sampled context c.

Here, $\hat{A} = (r - \hat{b}(x))/\hat{\sigma}(x)$ is the estimated group-wise advantage following GRPO (Shao et al., 2024a), and ϵ is the clipping hyperparameter. This formulation makes it explicit that LANPO is an orthogonal enhancement to specific policy gradient methods: we do not change the surrogate objective from an algorithmic perspective. Instead, we guide exploration by averaging the same objective over an evolving distribution of language contexts induced by the online-updated pool $\mathcal{E}_t$.

4.3. Feedback-Driven Inference at Test Time

Finally, the mechanisms that enabled stable training also equip LANPO models with reinforced skills to leverage language feedback at test-time. Given a problem, the policy can: **Solve from scratch** with no external context, preserving robustness and efficiency. **Leverage feedback** by (i) retrieving relevant entries from the final experience pool $\mathcal{E}_T$ accumulated during training for inter-sample guidance, or (ii) applying the intra-sample reflection loop to its existing attempts at the problem.

Crucially, because the experience pool contains distilled, transferable lessons rather than raw solutions, and the self-reflection mechanism is label-free, LANPO naturally extends these benefits to the test phase. Explicitly enhanced by numerical rewards during training, these skills improve model performance without requiring external resources during deployment.

5. Experiments

We now turn to a detailed empirical study of LANPO. Our experiments examine its overall effectiveness, the roles of different design components, and the training dynamics that shed light on why it works. We briefly summarize our setup below and defer full details, including hyperparameters, to Appendix B.2.

Models and datasets. We evaluate three base models: Qwen3-4B, Qwen2.5-7B and Qwen3-14B (Yang et al., 2024a; 2025a), none of which are instruction-tuned. For math RL, we adopt the DAPO dataset (Yu et al., 2025) containing 17K competition-level problems. Performance is measured on AIME25, AIME24, AMC23, and MATH-500 (Hendrycks et al., 2021). When it comes to coding RL, we train models on the Eurus-2 dataset (Cui et al., 2025) with 26K samples. We present mean@32 accuracy following common evaluation configurations.

Training protocols. RL training is performed with the VeRL framework (Sheng et al., 2024), using GRPO (Shao et al., 2024a) with group size 16 as the policy loss. We adopt the clip-higher trick (Yu et al., 2025) with $\epsilon_{\text{low}} = 0.2$, $\epsilon_{\text{high}} = 0.28$. Prompts are truncated at 3,072 tokens and generations at 8,192 tokens. Lastly, the number of RL training steps is set to 330 (10 epochs) by default.

LANPO configuration. We pre-compute the relatedness scores between problems before training using Qwen2.5-7B to avoid online computation, and set the similarity threshold γ to be 0.9, details for which are presented in Appendix C.1. Unless otherwise stated, the feedback ratio p is 0.75. An initial SFT stage with 3K QA pairs from DeepSeek-V3 (DeepSeek-AI et al., 2025) equips models with summarization, reflection, and response skills before RL training. We provide the complete list of prompts that we use for LANPO in Appendix E.

5.1. Benchmark Performance

Zero-Shot Inference Performance. Table 1 presents the main comparison between GRPO and LANPO on mathematical reasoning benchmarks. On Qwen2.5-7B, GRPO achieves an average accuracy of 42.79, whereas LANPO consistently pushes this higher: intra-sample feedback reaches 44.27, inter-sample feedback 44.68, and combining both yields 44.09. The gains extend to Qwen3-14B as well, where LANPO with both kinds of feedback improves the average accuracy to 64.54 compared to 62.27 for GRPO. Table 2 provides results on competitive coding benchmarks, from which we observe that LANPO achieves +1.12% to +3.92% absolute improvements on the Codeforces evaluation set and +0.93% to +1.72% on average.

These improvements demonstrate that LANPO systematically enhances the policy's ability to solve problems without additional hints at test time. By making feedback leakage-free and collapse-resistant during training, the resulting policy internalizes more reusable reasoning strategies, which translates into stronger zero-shot performance.

Experience Augmented Inference. LANPO-trained models also benefit from explicit feedback at inference. As Table 1 shows, conducting an additional self-reflection step further boosts Qwen2.5-7B from 44.09 (zero-shot with both feedback pathways) to 45.06, and Qwen3-14B from 64.54 to 68.27. Similarly, retrieving the accumulated training-time experiences also promises another 1-2 points increase in accuracy for both models. When it comes to coding benchmarks, retrieving textual experiences accumulated from training also exhibits notable performance boost, elevating average accuracy for 2.32% and 2.82% for LANPO w/ Inter and LANPO w/ both respectively.

In summary, these results confirm that LANPO not only

Table 1. Performance comparison between GRPO and LANPO under different inference modes. For AIME 25, AIME 24, and AMC, results are reported as mean ± standard deviation over 32 test runs. The highest score for each metric within each model group is highlighted in **bold**.

Model	Training Configuration	Inference Mode	AIME 25	AIME 24	AMC	MATH	Avg
Qwen2.5-7B-Base-SFT	No RL	Zero-shot	$4.27_{\pm 3.84}$	$7.19_{\pm 2.86}$	$29.82_{\pm 3.30}$	59.00	25.07
	GRPO	Zero-shot	$13.85_{\pm 3.97}$	$17.81_{\pm 4.08}$	$58.51_{\pm 2.37}$	81.00	42.79
	LANPO w/ Intra	Zero-shot	$15.62_{\pm 3.67}$	$20.94_{\pm 4.00}$	$58.51_{\pm 2.80}$	82.00	44.27
		w/ Self-reflection	$\mathbf{17.40}_{\pm 3.50}$	$\mathbf{23.54}_{\pm 3.27}$	$58.85_{\pm 2.91}$	**83.40**	45.80
	LANPO w/ Inter	Zero-shot	$17.19_{\pm 3.37}$	$18.23_{\pm 4.01}$	$60.50_{\pm 2.58}$	82.80	44.68
		w/ Retrieval	$\mathbf{17.81}_{\pm 3.96}$	$20.00_{\pm 3.44}$	$\mathbf{69.28}_{\pm 2.43}$	82.40	**47.37**
	LANPO w/ Both	Zero-shot	$16.25_{\pm 4.28}$	$20.62_{\pm 4.16}$	$59.30_{\pm 3.12}$	80.20	44.09
		w/ Retrieval	$17.60_{\pm 4.20}$	$21.46_{\pm 3.95}$	$60.35_{\pm 2.56}$	81.40	45.20
		w/ Self-reflection	$\mathbf{17.81}_{\pm 4.60}$	$20.94_{\pm 3.43}$	$60.50_{\pm 2.33}$	81.00	45.06
Qwen3-14B-Base-SFT	No RL	Zero-shot	$15.42_{\pm 4.42}$	$18.02_{\pm 5.06}$	$55.87_{\pm 3.13}$	82.20	42.88
	GRPO	Zero-shot	$34.38_{\pm 4.93}$	$45.00_{\pm 5.49}$	$78.69_{\pm 2.45}$	91.00	62.27
	LANPO w/ Intra	Zero-shot	$38.23_{\pm 5.14}$	$46.15_{\pm 4.33}$	$79.89_{\pm 2.10}$	91.80	64.02
		w/ Self-reflection	$\mathbf{42.29}_{\pm 5.00}$	$53.75_{\pm 5.30}$	$82.49_{\pm 2.20}$	92.80	67.83
	LANPO w/ Inter	Zero-shot	$35.42_{\pm 3.51}$	$49.38_{\pm 6.20}$	$79.03_{\pm 2.35}$	92.60	64.11
		w/ Retrieval	$35.83_{\pm 4.93}$	$49.38_{\pm 6.48}$	$81.89_{\pm 2.01}$	**93.20**	65.08
	LANPO w/ Both	Zero-shot	$38.33_{\pm 4.17}$	$49.27_{\pm 5.99}$	$79.37_{\pm 2.48}$	91.20	64.54
		w/ Retrieval	$38.85_{\pm 5.11}$	$49.90_{\pm 5.62}$	$84.07_{\pm 2.50}$	90.60	65.86
		w/ Self-reflection	$41.46_{\pm 4.24}$	$\mathbf{55.00}_{\pm 4.71}$	$\mathbf{84.83}_{\pm 2.06}$	91.80	**68.27**

Table 2. Performance comparison between GRPO and LANPO on coding benchmarks. The base model is Qwen2.5-7B-Instruct fine-tuned on our SFT dataset. The best score for each metric is highlighted in **bold**.

Configuration	Codeforces	Taco	Codecontest	Avg.
No RL	$10.32_{\pm 1.74}$	$7.22_{\pm 0.77}$	$14.16_{\pm 0.94}$	10.57
GRPO	$41.51_{\pm 1.98}$	$24.57_{\pm 0.73}$	$40.00_{\pm 1.21}$	35.36
LANPO w/ Intra	$42.63_{\pm 2.13}$	$25.26_{\pm 0.82}$	$40.98_{\pm 0.62}$	36.29
w/ Reflect.	$42.28_{\pm 1.75}$	$25.23_{\pm 0.61}$	$41.90_{\pm 0.61}$	36.47
LANPO w/ Inter	$42.99_{\pm 1.86}$	$24.93_{\pm 0.66}$	$39.79_{\pm 0.95}$	35.90
w/ Retri.	$45.78_{\pm 1.28}$	$26.51_{\pm 0.70}$	$42.37_{\pm 0.87}$	38.22
LANPO w/ Both	$45.43_{\pm 1.37}$	$25.07_{\pm 0.94}$	$40.73_{\pm 0.81}$	37.08
w/ Retri.	$\mathbf{47.56}_{\pm 1.29}$	$\mathbf{27.60}_{\pm 0.98}$	$\mathbf{44.61}_{\pm 0.83}$	**39.92**
w/ Reflect.	$45.27_{\pm 1.67}$	$24.90_{\pm 0.91}$	$40.60_{\pm 0.89}$	36.92

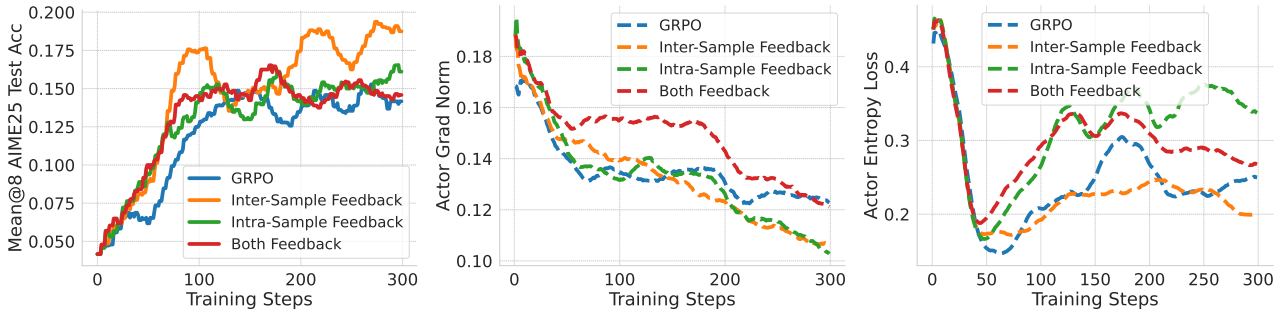
Table 3. **Effectiveness of training-time relevance filtering for inter-sample feedback.** We compare policies trained (i) without filtering ($\gamma = 0$) and (ii) with relevance filtering ($\gamma = 0.9$) when incorporating inter-sample feedback. Filtering makes retrieval consistently more beneficial (larger Δ_{retr}) and achieves the best overall performance.

Method	AIME 25	AIME 24	AMC	MATH	Avg
GRPO					
Zero-shot	13.85	17.81	58.51	81.00	42.79
w/ Retrieval	10.73	13.33	61.78	72.40	39.56
LANPO w/o relevance filtering					
Zero-shot	15.62	**20.31**	65.14	81.20	45.57
w/ Retrieval	15.94	17.29	**70.60**	81.20	46.26
LANPO w/ relevance filtering					
Zero-shot	17.19	18.23	60.50	**82.80**	44.68
w/ Retrieval	**17.81**	20.00	69.28	82.40	**47.37**

yields stronger stand-alone solvers but also equips them with the ability to reuse experiences dynamically at test time. In other words, the feedback-driven inference behavior that motivated our method emerges naturally as a byproduct of training. The results for Qwen3-4B are present in Appendix A.6 respectively, where similar improvements over GRPO are also consistently observed. In addition, we provide exemplar responses from models trained with LANPO in Appendix D.

5.2. Empirical Understandings

Training Dynamics. To understand how LANPO alters the learning process, we inspect the training dynamics on



(a) AIME25 test accuracy. *(b)* Gradient norm. *(c)* Action Entropy.

Figure 5. **Training dynamics.** We plot key metrics for LANPO variants and the GRPO baseline on Qwen2.5-7B. **(a)** Mean@8 test accuracy on the AIME25 benchmark, where both feedback mechanisms improve performance throughout training process. **(b)** Actor gradient norm, showing stable optimization of LANPO. **(c)** Action Entropy: The feedback mechanisms have distinct effects on exploration. Intra-sample feedback maintains high entropy, while inter-sample feedback reduces entropy.

Table 4. **Influence of the feedback ratio (p) on model performance.** A moderate ratio of $p = 0.75$ achieves the optimal balance between leveraging past experiences and preserving generalization, resulting in the highest average accuracy.

Setting	AIME 25	AIME 24	AMC	MATH	Avg
Ratio $p = 0.0$ (GRPO)					
Zero-shot	13.85	17.81	58.51	81.00	42.79
w/ Self-reflection	15.62	21.56	58.58	80.60	44.09
Ratio $p = 0.25$					
Zero-shot	15.62	18.96	57.79	79.20	42.89
w/ Self-reflection	**18.02**	19.90	58.47	82.40	44.70
Ratio $p = 0.50$					
Zero-shot	12.60	20.00	62.05	79.80	43.61
w/ Self-reflection	14.48	21.04	**62.50**	80.60	44.66
Ratio $p = 0.75$					
Zero-shot	15.62	20.94	58.51	82.00	44.27
w/ Self-reflection	17.40	**23.54**	58.85	**83.40**	**45.80**
Ratio $p = 1.00$					
Zero-shot	14.06	16.35	56.29	81.80	42.13
w/ Self-reflection	16.04	21.15	60.24	82.20	44.91

the AIME25 benchmark, as illustrated in Figure 5. Both intra- and inter-sample feedback mechanisms clearly outperform the GRPO baseline, achieving faster convergence and a higher final test accuracy (Figure 5a). These performance gains are realized without compromising stability, as evidenced by the smooth decline in the actor gradient norm for all methods (Figure 5b).

The plot of action entropy in Figure 5c reveals how our feedback mechanisms distinctly shape exploration. Intra-sample feedback, driven by Reward-Agnostic Reflection, sustains the highest entropy. This suggests the policy is encouraged to deviate from familiar patterns, thereby broadening its exploration. Conversely, inter-sample feedback, guided by Relevant Abstraction, produces the lowest entropy. This indicates that providing targeted principles effectively prunes the search space, focusing the policy's exploration.

In summary, these dynamics confirm that LANPO effectively uses language to guide the learning process: reflection prevents the policy from repeating familiar strategies, while abstraction provides strong heuristics to accelerate progress. We provide more visualization of training curves in Appendix A.5.

Ablation: Necessity of Filtering. We ablate the *training-time* relevance filtering used in inter-sample feedback by setting the similarity threshold γ to 0 (no filtering) versus 0.9, while keeping the remaining training pipeline unchanged. Table 3 reports both zero-shot performance and feedback-augmented inference for the resulting policies. The key quantity is the *relative gain*, i.e., $\Delta_{\text{retr}} = \text{Acc}(\text{w/ Retrieval}) - \text{Acc}(\text{Zero-shot})$, which reflects how much the policy can benefit from the provided experience. Without filtering, retrieval can be harmful: on AIME24, $\Delta_{\text{retr}} = -3.02$ (20.31→17.29), indicating that the policy is not reliably steered by the retrieved experience and is prone to behavior collapse. In contrast, training with filtering flips the retrieval gain on AIME24 to $+1.77$ (18.23→20.00) and yields the best overall retrieval-augmented performance (Avg 47.37). Beyond accuracy, we measure context sensitivity via the normalized edit distance Δ_{NED} between zero-shot and retrieval-conditioned outputs (Table 5, Appendix A.1). Filtering generally increases Δ_{NED}, suggesting reduced invariance to the injected experience and improved ability to utilize inter-sample feedback.

Ablation: Effect of Feedback Ratio. Table 4 proceeds by studying the probability p of including feedback during LANPO training. We adopt intra-sample feedback for this ablation. The results reveal that a low value ($p = 0.25$) underutilizes past experiences, performing similarly to standard GRPO. On the other hand, very high value ($p = 1.0$) results in the lowest zero-shot performance on average (42.13) due to increased training instability. The best trade-off arises at moderate ratios: $p = 0.75$ achieves the highest av-

erage accuracy (45.80 with self-reflection), outperforming both lower and higher ratios. This confirms that balancing feedback-aware and from-scratch rollouts is key to preserving generalization.

Training overhead: LANPO introduces computational overhead from three primary sources: (1) managing the experience pool, (2) processing longer sequences for feedback-guided rollouts, and (3) an auxiliary summarization stage. The cost of experience pool management is minimal, as these operations are lightweight CPU operations. The primary overhead arises from increased sequence lengths. Both inter/intra-sample expand the model's input and output length, directly increasing the processing time per step. Additionally, the summarization step requires a separate generation process. Accounting for the increased training time, we compare GRPO and LANPO under the same training time. Results show that LANPO still prevails in the challenging AIME-24/25 benchmarks. The detailed breakdown of training time and performance comparison between GRPO and LANPO under the same budget are provided in Appendix A.7. In addition, we note that there are many ways to accelerate LANPO training, such as overlapping the CPU/GPU operations, condensing the prompts, and conducting the summarization stage asynchronously, which we leave for future work.

6. Discussion and Conclusion

To summarize, this work demonstrates that the rollouts generated during LLM RL training can be harnessed as language feedback to reliably improve sample efficiency when paired with safeguards that prevent leakage and collapse. By separating the roles of signals—using reward-agnostic reflection to support intra-sample refinement and relevant abstraction to enable inter-sample transfer—LANPO turns prior rollouts into structured guidance for exploration, while numerical rewards determine what the policy ultimately learns. The resulting self-contained training pipeline yields consistent gains across models and benchmarks over GRPO, improves zero-shot performance, and enables experience-driven inference.

Beyond empirical improvements, LANPO offers a practical blueprint for integrating linguistic structure into policy optimization in LLMs: curate distilled experiences from trial-and-error, enforce relevance, and promote reflective reasoning under a stable policy gradient objective. These principles scale across settings and model sizes, opening a path toward more sample-efficient, robust, and adaptable RL for reasoning tasks. Future work may explore applying LANPO to non-verifiable tasks, designing explicit credit assignment between feedback and solution, and coordinating the experience pool during RL training in more efficient ways.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Agrawal, L. A., Tan, S., Soylu, D., Ziems, N., Khare, R., Opsahl-Ong, K., Singhvi, A., Shandilya, H., Ryan, M. J., Jiang, M., et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*, 2025.

Ankner, Z., Liu, B., Sun, M., et al. Critique-out-loud reward models. *arXiv preprint arXiv:2408.11791*, 2024.

Bai, Y., Jones, A., Ndousse, K., Askell, A., Chen, A., et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

Baronio, C., Marsella, P., Pan, B., Guo, S., and Alberti, S. Kevin: Multi-turn rl for generating cuda kernels. 2025.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *NeurIPS*, 33:1877–1901, 2020.

Chen, A., Scheurer, J., Campos, J. A., Korbak, T., Chan, J. S., Bowman, S. R., Cho, K., and Perez, E. Learning from natural language feedback. *Transactions on Machine Learning Research*, 2024.

Cui, G., Yuan, L., Wang, Z., Wang, H., Li, W., He, B., Fan, Y., Yu, T., Xu, Q., Chen, W., et al. Process reinforcement through implicit rewards. *arXiv preprint arXiv:2502.01456*, 2025.

DeepSeek-AI, Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., Zhao, C., Deng, C., Zhang, C., Ruan, C., Dai, D., Guo, D., Yang, D., Chen, D., Ji, D., Li, E., Lin, F., Dai, F., Luo, F., Hao, G., Chen, G., Li, G., Zhang, H., Bao, H., Xu, H., Wang, H., Zhang, H., Ding, H., Xin, H., Gao, H., Li, H., Qu, H., Cai, J. L., Liang, J., Guo, J., Ni, J., Li, J., Wang, J., Chen, J., Chen, J., Yuan, J., Qiu, J., Li, J., Song, J., Dong, K., Hu, K., Gao, K., Guan, K., Huang, K., Yu, K., Wang, L., Zhang, L., Xu, L., Xia, L., Zhao, L., Wang, L., Zhang, L., Li, M., Wang, M., Zhang, M., Zhang, M., Tang, M., Li, M., Tian, N., Huang, P., Wang, P., Zhang, P., Wang, Q., Zhu, Q., Chen, Q., Du, Q., Chen, R. J., Jin, R. L., Ge, R., Zhang, R., Pan, R., Wang, R., Xu, R., Zhang, R., Chen, R., Li, S. S., Lu, S., Zhou, S., Chen, S., Wu, S., Ye, S., Ye, S., Ma, S., Wang, S., Zhou, S., Yu, S., Zhou, S., Pan, S., Wang, T., Yun, T., Pei, T., Sun, T., Xiao, W. L., Zeng, W., Zhao, W., An, W., Liu, W., Liang, W., Gao, W., Yu, W., Zhang, W., Li, X. Q., Jin, X., Wang, X., Bi, X., Liu, X., Wang, X., Shen, X., Chen, X., Zhang, X., Chen, X., Nie, X., Sun, X., Wang, X., Cheng, X., Liu, X., Xie, X., Liu, X., Yu, X., Song, X., Shan, X., Zhou, X., Yang, X., Li, X., Su, X., Lin, X., Li, Y. K., Wang, Y. Q., Wei, Y. X., Zhu, Y. X., Zhang, Y., Xu, Y., Xu, Y., Huang, Y., Li, Y., Zhao, Y., Sun, Y., Li, Y., Wang, Y., Yu, Y., Zheng, Y., Zhang, Y., Shi, Y., Xiong, Y., He, Y., Tang, Y., Piao, Y., Wang, Y., Tan, Y., Ma, Y., Liu, Y., Guo, Y., Wu, Y., Ou, Y., Zhu, Y., Wang, Y., Gong, Y., Zou, Y., He, Y., Zha, Y., Xiong, Y., Ma, Y., Yan, Y., Luo, Y., You, Y., Liu, Y., Zhou, Y., Wu, Z. F., Ren, Z. Z., Ren, Z., Sha, Z., Fu, Z., Xu, Z., Huang, Z., Zhang, Z., Xie, Z., Zhang, Z., Hao, Z., Gou, Z., Ma, Z., Yan, Z., Shao, Z., Xu, Z., Wu, Z., Zhang, Z., Li, Z., Gu, Z., Zhu, Z., Liu, Z., Li, Z., Xie, Z., Song, Z., Gao, Z., and Pan, Z. Deepseek-v3 technical report. 2025.

Gu, J., Jiang, X., Shi, Z., Tan, H., Zhai, X., Xu, C., Li, W., Shen, Y., Ma, S., Liu, H., et al. A survey on llm-as-a-judge. *The Innovation*, 2024.

Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

He, J., Li, T., Feng, E., Du, D., Liu, Q., Liu, T., Xia, Y., and Chen, H. History rhymes: Accelerating llm reinforcement learning with rhymerl. 2025.

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. *arXiv preprint arXiv:2103.03874*, 2021.

Kumar, A., Zhuang, V., Agarwal, R., Su, Y., Co-Reyes, J. D., Singh, A., Baumli, K., Iqbal, S., Bishop, C., Roelofs, R., Zhang, L. M., McKinney, K., Shrivastava, D., Paduraru, C., Tucker, G., Precup, D., Behbahani, F., and Faust, A. Training language models to self-correct via reinforcement learning. 2024.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., tau Yih, W., Rocktäschel, T., Riedel, S., and Kiela, D. Retrieval-augmented generation for knowledge-intensive nlp tasks. 2021.

Li, X., Sun, X., Li, J., Wang, A., and Shum, C. Cuda-l1: Improving cuda optimization via contrastive reinforcement learning. 2025.

Madaan, A., Tandon, N., and Others. Memory-assisted prompt editing to improve gpt-3 after deployment. In *EMNLP*, 2022.

Madaan, A., Tandon, N., Gupta, P., et al. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*, 2023.

OpenAI. Learning to reason with llms, 2024. URL https://openai.com/index/learning-to-reason-with-llms/.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., et al. Training language models to follow instructions with human feedback. In *NeurIPS*, 2022.

Qu, Y., Setlur, A., Smith, V., Salakhutdinov, R., and Kumar, A. Pope: Learning to reason on hard problems via privileged on-policy exploration, 2026.

Rafailov, R. et al. Direct preference optimization: Your language model is secretly a reward model. In *NeurIPS*, 2023.

Robertson, S. and Zaragoza, H. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends in Information Retrieval*, 2009.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017a.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017b.

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Zhang, M., Li, Y., Wu, Y., and Guo, D. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024a.

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Zhang, M., Li, Y., Wu, Y., and Guo, D. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024b.

Sheng, G., Zhang, C., Ye, Z., Wu, X., Zhang, W., Zhang, R., Peng, Y., Lin, H., and Wu, C. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv:2409.19256*, 2024.

Shinn, N., Cassano, F., Berman, E., Gopinath, A., Narasimhan, K., and Yao, S. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023.

Wang, G., Xie, Y., Jiang, Y., et al. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Wang, J., Wang, Z., Yang, L., Zhou, H., Shao, K., Guo, S., Chen, Y., Zhang, G., Yan, X., Lee, K. H., and Lee, K. Y. Agentfly: Fine-tuning llm agents without fine-tuning llms. 2025.

Wang, Y., Wu, Y., Wei, Z., Jegelka, S., and Wang, Y. A theoretical understanding of self-correction through in-context alignment. In *NeurIPS 2024*, 2024a.

Wang, Y., Wu, Y., Wei, Z., Jegelka, S., and Wang, Y. A theoretical understanding of self-correction through in-context alignment. *NeurIPS 2024*, 2024b.

Yang, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Li, C., Liu, D., Huang, F., Wei, H., et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024a.

Yang, A., Li, A., Yang, B., Zhang, B., Hui, B., Zheng, B., Yu, B., Gao, C., Huang, C., Lv, C., Zheng, C., Liu, D., Zhou, F., Huang, F., Hu, F., Ge, H., Wei, H., Lin, H., Tang, J., Yang, J., Tu, J., Zhang, J., Yang, J., Yang, J., Zhou, J., Zhou, J., Lin, J., Dang, K., Bao, K., Yang, K., Yu, L., Deng, L., Li, M., Xue, M., Li, M., Zhang, P., Wang, P., Zhu, Q., Men, R., Gao, R., Liu, S., Luo, S., Li, T., Tang, T., Yin, W., Ren, X., Wang, X., Zhang, X., Ren, X., Fan, Y., Su, Y., Zhang, Y., Zhang, Y., Wan, Y., Liu, Y., Wang, Z., Cui, Z., Zhang, Z., Zhou, Z., and Qiu, Z. Qwen3 technical report. 2025a.

Yang, L., Yu, Z., Zhang, T., Cao, S., Xu, M., Zhang, W., Gonzalez, J. E., and Cui, B. Buffer of thoughts: Thought-augmented reasoning with large language models. 2024b.

Yang, L., Yu, Z., Cui, B., and Wang, M. Reasonflux: Hierarchical llm reasoning via scaling thought templates. 2025b.

Yu, Q., Zhang, Z., Zhu, R., Yuan, Y., Zuo, X., Yue, Y., Dai, W., Fan, T., Liu, G., Liu, L., Liu, X., Lin, H., Lin, Z., Ma, B., Sheng, G., Tong, Y., Zhang, C., Zhang, M., Zhang, W., Zhu, H., Zhu, J., Chen, J., Chen, J., Wang, C., Yu, H., Song, Y., Wei, X., Zhou, H., Liu, J., Ma, W.-Y., Zhang, Y.-Q., Yan, L., Qiao, M., Wu, Y., and Wang, M. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Yuan, W., Pang, R. Y., Cho, K., Li, X., Sukhbaatar, S., Xu, J., and Weston, J. Self-rewarding language models. *arXiv preprint arXiv:2401.10020*, 2024.

Yuan, Y. and Xie, T. Reinforce LLM reasoning through multi-agent reflection. In *ICML 2024*, 2025.

Zheng, L., Chiang, W.-L., Sheng, Y., et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *arXiv preprint arXiv:2306.05685*, 2023.

Ziegler, D. M., Stiennon, N., Wu, J., Brown, T., Radford, A., Amodei, D., Christiano, P., and Irving, G. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

A. Additional Results

A.1. Discussion on Behavior Collapse

In our setting, *behavior collapse* describes a failure mode where the policy under-utilizes (or entirely ignores) additional, potentially helpful information provided in the context—such as a hint, a worked solution to a related problem, or a previously verified strategy—and instead responds as if that information were absent. Importantly, this is not merely occasional irrelevance: we observe systematic insensitivity to the added experience even when it encodes a transferable method.

Formalization. Let x denote the target problem and c the *experience* (e.g., a correct trajectory from another sample) inserted into the model's context. Let $\pi_\theta(\cdot \mid x, c)$ be the trained policy with experience and $\pi_\theta(\cdot \mid x)$ the same policy without it. We characterize the *influence* of c on the model by a distance measure:

$$\mathcal{I}(x, c) \;:=\; \Delta\Big(\pi_\theta(\cdot \mid x, c),\; \pi_\theta(\cdot \mid x)\Big), \tag{4}$$

where $\Delta(\cdot, \cdot)$ can be any similarity/distance defined over outputs or distributions (e.g., KL/JSD between token distributions, embedding distance, or string-level distance between decoded responses). A model exhibits *collapse* if the influence is small with high probability:

$$p_{\text{collapse}}(\varepsilon) \;:=\; \Pr_{(x,c)}\Big[\mathcal{I}(x, c) \le \varepsilon\Big], \tag{5}$$

for a small ε. Intuitively, a large p_{collapse} means the policy becomes nearly invariant to the presence of experience.

In our experiments, we instantiate Δ using a *normalized edit distance* between decoded responses:

$$\Delta_{\text{NED}}(x, c) := \frac{\text{ED}(y_{x,c},\, y_x)}{\max\{|y_{x,c}|, |y_x|\}}, \tag{6}$$

where $y_{x,c} = \text{decode}(\pi_\theta(\cdot \mid x, c))$ and $y_x = \text{decode}(\pi_\theta(\cdot \mid x))$. A small Δ_{NED} indicates that c has weak behavioral effect (collapse).

An example of behavior collapse. We provide a representative example in Example A.1. The model correctly notices superficial differences between the Input Problem and the provided Experience, but it then prematurely concludes that the experience is irrelevant and proceeds with essentially the same solution behavior as in the no-experience setting.

Alleviation. This motivates the *relevance filtering* step in relevant abstraction (Section 4.1), which increases the expected utility of c by restricting retrieval to sufficiently related problems. To isolate its effect, we compare LANPO trained with filtering enabled to an ablation that disables filtering (setting $\gamma = 0$ while keeping the rest unchanged). Table 5 reports Δ_{NED} between experience-conditioned responses and their zero-shot counterparts. Enabling filtering consistently increases Δ_{NED}, indicating stronger behavioral dependence on retrieved experience (reduced collapse). This aligns with the accuracy gains reported in our main results, suggesting that the increased dependence reflects more substantive utilization rather than purely incidental perturbations.

Table 5. Effect of relevance filtering on behavior collapse. We report the character-level mean normalized/absolute edit distance Δ_{NED} between retrieval-conditioned responses and their zero-shot counterparts (same deterministic decoding). Larger values indicate stronger behavioral dependence on retrieved experience, suggesting reduced context insensitivity.

Model	AIME-25	AIME-24	AMC	MATH-500
LANPO Without Filtering	0.695 (7038)	0.712 (9482)	0.680 (5759)	0.660 (3699)
LANPO With Filtering	0.720 (10287)	0.714 (11218)	0.704 (6716)	0.690 (4268)

Example for Behavior Collapse

Input Problem: In triangle ABC points D and E lie on $\overline{AB}$ so that AD $<$ AE $<$ AB, while points F and G lie on $\overline{AC}$ so that AF $<$ AG ¡ AC. Suppose AD = 4, DE = 16, EB = 8, AF = 13, FG = 52, and GC = 26. Let M be the reflection of D through F, and let N be the reflection of G through E. The area of quadrilateral DEGF is 288. Find

the area of heptagon AFNBCEM

Experience

Problem: On $\triangle ABC$ let D be a point on side $\overline{AB}$, F be a point on side $\overline{AC}$, and E be a point inside the triangle so that $\overline{DE} \parallel \overline{AC}$ and $\overline{EF} \parallel \overline{AB}$. Given that AF = 6, AC = 33, AD = 7, AB = 26, and the area of quadrilateral ADEF is 14, find the area of $\triangle ABC$

Flow of thought

1. **Understanding the Problem Setup**: The problem presents us with a triangle ABC with given segment lengths and parallel lines. The initial steps involve identifying what each parallel line does to create similar triangles.
2. **Visualizing Parallel Lines and Similar Triangles**: ...

Takeaways

1. **Visualize the Problem**: Always visualize the problem setup to identify parallel lines and similar triangles.
2. **Similarity Ratios**: ...
3. **Segment Proportions**: ...
4. **Quadrilateral Area and Proportions**: ...
5. **Final Area Calculation**: ...

Response from a collapsed model

- **Analysis**: The provided experience outlines a structured approach to solving geometric problems involving parallel lines, similarity of triangles, and area calculations. **However, the current problem is different in its setup and details. The problem involves reflections and specific segment lengths, which are not directly addressed by the previous experience.**
- **Decision**: Since the current problem is different in its setup and requires a different approach, I will create a new plan tailored specifically to this problem. The previous experience is not directly applicable here.

A.2. Additional Analysis on Naive Intra-Sample Feedback

In Section 3, we show that directly using ground-truth labels as intra-sample feedback can hinder the policy from acquiring generalizable reasoning skills. Here, we further examine a less extreme variant of such feedback: exposing only a prefix of the reference solution. This setting allows us to test whether partial ground-truth reasoning traces can provide useful guidance without fully revealing the answer.

We fine-tune Qwen2.5-7B-Instruct (Yang et al., 2024a) on the MATH training set (Hendrycks et al., 2021). For each reference solution with length L, we prepend the first $\lfloor pL \rfloor$ tokens to the prompt as a hint, where $p \in \{0.00, 0.25, 0.50, 0.75\}$. The ratio $p = 0.00$ corresponds to the no-leakage baseline. Full results are presented in Table 6.

A.3. Additional Analysis on Naive Intra-Sample Feedback

In Section 3, we show that directly using ground-truth labels as intra-sample feedback can hinder the policy from acquiring generalizable reasoning skills. Here, we further examine a less extreme variant of such feedback: exposing only a prefix of the reference solution. This setting allows us to test whether partial ground-truth reasoning traces can provide useful guidance without fully revealing the answer. Our experiments are conducted on the MATH-500 training set (Hendrycks et al., 2021) with Qwen2.5-7B (Yang et al., 2024a). Specifically, for each reference solution with length L, we prepend the first $\lfloor pL \rfloor$ tokens to the prompt as a hint, where $p \in \{0.00, 0.25, 0.50, 0.75\}$. The ratio $p = 0.00$ corresponds to the no-leakage baseline. Full results are presented in Table 6.

The results show that leaking partial reference solutions does not lead to consistent improvement over the no-leakage

Table 6. Additional study on intra-sample feedback via partial reference-solution leakage. We fine-tune Qwen2.5-7B-Instruct on the MATH training set with three independent runs, varying the fraction of leaked reference-solution tokens for difficult problems, i.e., Level 4 and Level 5. For AIME24, AIME25, and AMC, each run reports the mean $\pm$ standard deviation over 32 evaluation trials. The best averaged result in each column is highlighted in bold.

Ratio	Run	AIME25	AIME24	AMC	MATH500	Avg.
0.00	Run 1	11.25 ± 4.23	14.37 ± 3.27	46.57 ± 2.95	78.20	37.60
	Run 2	10.21 ± 3.63	13.23 ± 4.04	44.62 ± 3.17	77.80	36.47
	Run 3	8.96 ± 4.37	14.37 ± 3.58	46.65 ± 3.06	78.80	37.20
	Avg.	**10.14** ± 1.15	**13.99** ± 0.66	45.95 ± 1.15	78.27 ± 0.50	37.09 ± 0.57
0.25	Run 1	9.48 ± 4.09	13.02 ± 3.16	47.03 ± 2.36	79.60	37.28
	Run 2	8.12 ± 4.40	12.50 ± 4.25	45.63 ± 2.59	78.00	36.06
	Run 3	9.58 ± 4.39	14.69 ± 3.32	45.07 ± 2.87	76.60	36.48
	Avg.	9.06 ± 0.82	13.40 ± 1.14	45.91 ± 1.01	78.07 ± 1.50	36.61 ± 0.62
0.50	Run 1	8.23 ± 4.25	14.69 ± 3.32	48.38 ± 3.13	79.40	37.68
	Run 2	9.79 ± 4.40	13.44 ± 3.86	46.01 ± 3.09	78.00	36.81
	Run 3	6.98 ± 4.67	13.65 ± 4.28	46.99 ± 3.12	79.60	36.81
	Avg.	8.33 ± 1.41	13.93 ± 0.67	**47.13** ± 1.19	**79.00** $\pm$ **0.87**	**37.10** ± 0.50
0.75	Run 1	10.31 ± 4.20	12.19 ± 2.71	45.59 ± 2.66	76.40	36.12
	Run 2	8.33 ± 4.33	13.23 ± 3.38	46.57 ± 2.84	77.40	36.38
	Run 3	7.92 ± 3.61	11.98 ± 4.48	45.26 ± 3.13	77.00	35.54
	Avg.	8.85 ± 1.28	12.47 ± 0.67	45.81 ± 0.68	76.93 ± 0.50	36.01 ± 0.43

baseline. Although the ratio $p = 0.50$ achieves a slightly higher average score, the gain is marginal and falls within the variance across repeated runs. Moreover, increasing the leakage ratio to $p = 0.75$ noticeably degrades performance, suggesting that stronger dependence on reference prefixes can hurt generalization.

We attribute this behavior to two factors. First, reference solutions are off-policy: they may follow reasoning paths that are not naturally induced by the current policy, making them difficult for the model to internalize as reusable reasoning strategies. Second, solution-prefix leakage introduces a train–test mismatch. During training, the model can condition on externally provided reasoning prefixes, whereas at test time it must initiate and develop the solution from scratch. As a result, the learned policy may become overly dependent on such prefixes rather than improving its own reasoning ability. It is notable that POPE (Qu et al., 2026) shows that only revealing partial solutions on challenging problems can bring additional training signals, thus benefiting learning. However, the method requires careful mixture of hard and easy problems to succeed, and the drastic mismatch between training and inference still exists.

By contrast, LANPO avoids these pitfalls by letting the policy reflect on its own trajectories without exposing ground-truth labels or reference solutions. This keeps the feedback on-policy and aligns the training process with the test-time setting, where the model must rely on its own generated reasoning.

A.4. More Evaluation Results

Table 7. Testing SFT & GRPO trained models with language feedback.

Model Configuration		AIME25	AIME24	AMC	MATH	**Average**
Model	**Feedback Type**	(%)	(%)	(%)	(%)	(%)
Qwen2.5-7B-base-SFT	w/ Retrieval	5.63	6.98	44.16	66.00	30.69
	w/ Self-reflection	4.17	8.54	33.25	62.20	27.04
RL w/ GRPO	w/ Retrieval	10.73	13.33	61.78	72.40	39.56
	w/ Self-reflection	15.62	21.56	58.58	80.60	44.09
Qwen3-14B-base-SFT	w/ Retrieval	17.08	19.69	65.81	83.60	46.55
	w/ Self-reflection	15.52	18.54	56.66	82.40	43.28
RL w/ GRPO	w/ Retrieval	28.85	37.29	80.69	90.40	59.31
	w/ Self-reflection	37.29	48.85	81.66	91.60	64.85

Table 7 presents the evaluation of SFT and GRPO models when language feedback is provided during inference. Two key

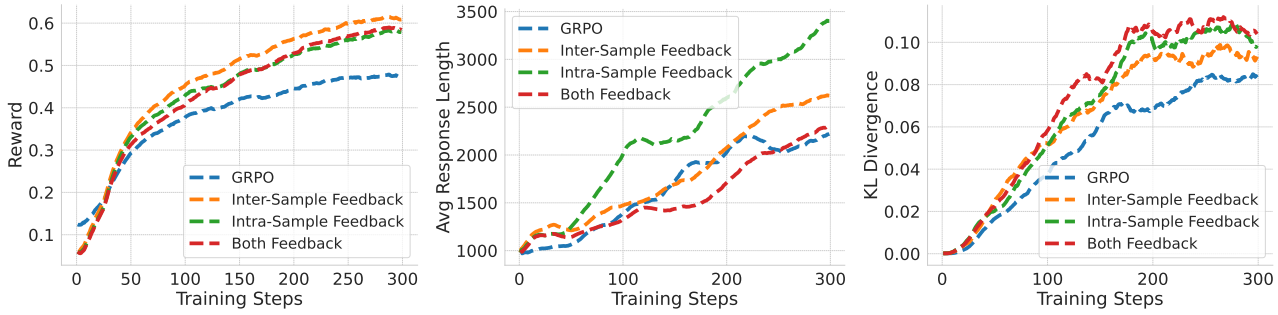
Table 8. Performance comparison between GRPO and LANPO on math benchmarks. The base model here is the Qwen3-4B-Base fine-tuned on our SFT dataset. The highest score for each metric within each model group is highlighted in **bold**.

Model	Training Configuration	Inference Mode	AIME 25	AIME 24	AMC	MATH	Avg
Qwen3-4B-Base-SFT	No RL	Zero-shot	$41.77_{\pm 4.49}$	$53.23_{\pm 4.37}$	$79.86_{\pm 2.49}$	94.40	67.32
	GRPO	Zero-shot	$52.50_{\pm 4.64}$	$59.90_{\pm 5.68}$	$87.80_{\pm 2.43}$	**96.60**	74.22
	LANPO w/ Intra	Zero-shot	$54.27_{\pm 4.95}$	$60.52_{\pm 5.28}$	$89.16_{\pm 1.57}$	95.80	74.94
		w/ Self-reflection	$54.79_{\pm 5.13}$	$61.04_{\pm 5.30}$	$89.27_{\pm 1.58}$	96.00	75.28
	LANPO w/ Inter	Zero-shot	$57.60_{\pm 3.75}$	$\mathbf{62.29_{\pm 4.60}}$	$89.34_{\pm 1.17}$	**96.60**	**76.46**
		w/ Retrieval	$56.67_{\pm 4.44}$	$61.04_{\pm 5.32}$	$\mathbf{90.17_{\pm 1.56}}$	94.60	75.62
	LANPO w/ Both	Zero-shot	$57.29_{\pm 5.43}$	$59.38_{\pm 5.92}$	$88.18_{\pm 2.22}$	96.40	75.31
		w/ Retrieval	$54.90_{\pm 4.71}$	$59.58_{\pm 5.19}$	$88.74_{\pm 1.92}$	95.00	74.56
		w/ Self-reflection	$\mathbf{58.12_{\pm 5.20}}$	$60.62_{\pm 5.92}$	$88.52_{\pm 2.06}$	**96.60**	75.97

findings emerge from the results. First, for models trained via Reinforcement Learning (RL) without exposure to language feedback, their performance at test time may not increase when such feedback is introduced. Second, despite notable gain in certain cases, performance of models trained with naive GRPO remains consistently lower than that of models explicitly trained with language feedback.

A.5. Training Curves

We present the curves for reward, average response length, and KL divergence during RL training in Figure 6.



(a) Reward *(b)* Average response length *(c)* KL divergence

Figure 6. **Training dynamics for LANPO and GRPO on Qwen2.5-7B.** The figure compares key metrics during reinforcement learning for LANPO variants against the GRPO baseline. We plot: (a) the training reward, (b) the average length of generated responses, and (c) the KL divergence from the reference policy. Notably, for the reward metric in (a), LANPO models receive a 0.1 bonus for responses that adhere to the correct format when language feedback is utilized.

A.6. Results on Qwen3-4B

LANPO's training process requires the policy model to generate summaries and reflect on its attempts, a mechanism that relies on stronger instruction-following capabilities than standard zero-shot reasoning. This motivates an investigation into whether smaller models possess sufficient capabilities to benefit from LANPO. We test this on Qwen3-4B-Instruct, a model with significantly fewer parameters. Following the procedure outlined in Section 4, we first fine-tune the model with SFT, and then conduct RL training using GRPO, and LANPO respectively.

The results, presented in Table 8, demonstrate that LANPO consistently outperforms the GRPO baseline across all four benchmarks. The performance gains are particularly notable on the more challenging AIME-24 (62.29 vs. 59.90) and AIME-25 (56.67 vs. 52.50) datasets. This outcome confirms that LANPO is effective at enhancing complex reasoning abilities, even for smaller-scale language models.

A.7. Analysis of Training Overhead

To analyze the trade-offs associated with integrating language feedback, we compare the computational overhead and performance of LANPO against GRPO. Table 9 provides a breakdown of the training time per step. The primary overhead for LANPO arises from processing the additional text generated for feedback (summaries and reflections), which increases the time required for calculating log probabilities and performing the weight update. Consequently, a single LANPO step takes approximately twice as long as a GRPO step.

Table 10 examines whether this increased per-step cost is justified by performance gains under a similar total compute budget. The results show that LANPO trained for 330 steps ($\sim$146 hours) achieves superior performance on both AIME benchmarks compared to GRPO trained for 730 steps ($\sim$160 hours). Despite GRPO processing more trajectories over more steps, LANPO's approach yields better results in a comparable timeframe, demonstrating its improved sample and computational efficiency.

Table 9. Breakdown of training time per step for GRPO vs. LANPO on Qwen2.5-7B. Measurements are an average from step 200.

Training Stage	**GRPO**	**LANPO (inter-sample)**	**LANPO (intra-sample)**
Rollout	222s (30.62%)	328s (23.05%)	346s (23.96%)
Calculating Old/Ref Log-Prob	150s (20.69%)	324s (22.77%)	314s (21.75%)
Reward Calculation	8s (1.10%)	8s (0.56%)	8s (0.55%)
Weight Update	342s (47.17%)	699s (49.12%)	708s (49.03%)
Other Overhead	3s (0.41%)	64s (4.50%)	68s (4.71%)
End to End	**725s**	**1423s**	**1444s**
Total Training Time (330 steps)	**~72 hours**	**~146 hours**	**~138 hours**

Table 10. Performance of Qwen2.5-7B with GRPO and LANPO under similar compute budgets. Best scores are in **bold**.

Configuration	**AIME-25**	**AIME-24**	**Total Training Time**	**Total Input Tokens**	**Total Output Tokens**
GRPO-330 steps	13.02	17.29	~72 hours	~0.04M	~0.5M
GRPO-730 steps	15.90	18.60	~160 hours	~0.08M	~1M
LANPO-330 steps	**16.04**	**19.48**	~146 hours	~0.2M	~0.6M

B. Experimental Details

B.1. SFT Training

The key hyperparameters and settings for this SFT process are outlined in Table 11.

Table 11. Supervised Fine-Tuning (SFT) Hyperparameters

Parameter	**Value**
Training Epochs	1
Global Batch Size	64
Per-GPU Micro-batch Size	1
Max Sequence Length	8192 tokens
Sequence Parallelism Size	2

B.2. RL Training

The core hyper-parameter for RL training are listed in Table 12, and parameters for data handling and generation are listed in Table 13. Lastly, we set the maximum number of feedback saved in each step to be 8, the maximum summary length to be 2048.

Table 12. Core Training Hyperparameters of RL

Parameter	**Value**
Policy Loss	PPO
Actor Learning Rate	1e-6
LR Schedule	Cosine with 10% warmup
PPO Clipping Range (Low, High)	[0.20, 0.28]
KL Divergence Loss Coefficient	0.0005
Reward Function	Correctness Reward Plus Format Reward
Total Training Epochs	10
PPO Mini-batch Size	64

Table 13. RL Generation Parameters

Parameter	**Value**
Max Prompt Length	3072 tokens
Max Response Length	8192 tokens
Rollout Samples per Prompt (n)	16
Validation Generation Settings	
Decoding Strategy	Sample
Temperature	0.6
Top-p	0.9
Validation Samples per Prompt (n)	8

B.3. Preliminary Study

Test-time Performance: For in-context examples, we always provide one correct example each time, while we only consider those incorrect responses for self-reflection. The inference configuration is same as Table 13.

RL Training: RL training is conducted in a similar but simplified training configuration of the main experiments, using the same model. Specifically, we reduce response length to 4096, group size n to be 8, and training epochs to be 5. For intra-sample feedback, we provide the historical trajactories accompanied by the ground truth answer. For inter-sample feedback, the selection procedure is the same as LANPO except for no filtering is applied.

C. Implementation Details

C.1. Calculation of Relevance Score

To quantify the relevance of a piece of language feedback to a given math problem, we employ a zero-shot classification approach using a Large Language Model (LLM). For a specific problem p and a candidate feedback c, we construct a structured prompt, detailed below, which directs the model to evaluate the feedback's utility and relevance. The LLM processes this prompt and generates output logits for the tokens `yes` and `no`, which we denote as l_y and l_n, respectively. The relevance score, $r(p, c)$, is then computed as the softmax probability of the "yes" token:

$$r(p, c) = \frac{\exp(l_y)}{\exp(l_y) + \exp(l_n)} \tag{7}$$

At test time, the corpus of candidate feedback is too large for an exhaustive evaluation of every option. To manage this

computational expense, we adopt a two-stage retrieval-and-rerank strategy. First, we utilize the BM25 algorithm (Robertson & Zaragoza, 2009) to efficiently retrieve the top-k most relevant feedback candidates. Subsequently, we apply our LLM-based scoring method (as defined in Equation (7)) to this smaller subset to rerank the candidates and identify the most helpful feedback.

Prompt for Relevance Estimation

```
#### Math Problem: {problem}.
#### Feedback: {feedback}.
```

Determine whether the language feedback above is closely relevant and helpful to the math problem. Carefully think about whether the feedback provides highly useful insights, information, or techniques in solving the problem. Consider inspecting specific details shown in the feedback and imagine how you would approach the problem using it.

```
#### Answer with yes or no.
#### Answer:
```

C.2. Experience Pool

Overview. The experience pool $\mathcal{E}$ is a lightweight, problem-indexed memory that stores distilled language experiences from past rollouts. Each stored item contains the original problem, response, and the corresponding *structured summary*. The structured summary consists of a high-level solution outline covering the major steps, and the transferable principles and pitfalls part compressing the key ideas/mistakes in the solution. Necessary metadata (e.g., reward, source, timestamp) is also included.

Organization and capacity. We maintain a separate queue of experiences for each problem x. For each x, the pool keeps at most 128 entries; when the capacity is exceeded, we evict the oldest entry using FIFO (First-In-First-Out). The pool is initialized as empty for every training run.

Retrieval usage. At both training and test time, the policy may sample a context $c \sim p_c(\cdot \mid x, \mathcal{E})$ from the pool. (i) For *intra-sample reflection*, p_c prioritizes the model's recent attempts on the same problem x, providing material to critique and revise without accessing gold labels. (ii) For *inter-sample guidance*, p_c retrieves entries from semantically related problems and applies relevance filtering; only entries with relevance above a threshold γ are included, ensuring the retrieved context is aligned and generalizable.

Test-time persistence and baseline evaluation. We keep using the final pool accumulated during training at test time. When evaluating SFT/GRPO models (which do not maintain native experience pools) under the inter-sample feedback setting, we reuse the pool accumulated by LANPO trained with inter-sample feedback to provide a matched retrieval source.

C.3. Weighted Sampling for Feedback Retrieval

We employ a weighted sampling strategy to prioritize higher-quality language feedback when constructing training batches. Feedback is first sorted by relevance score or time order, and a pool is created from the top candidates. Each feedback in this pool is assigned a weight using a linear decay function ($w_i \propto 1/(i+1)$), which is then normalized to form a probability distribution. We then sample groups of feedback from this pool without replacement, ensuring that higher-ranked items are more likely to be selected.

C.4. Format Reward

To encourage adherence to a specific output structure, we introduce a small format-based reward. The model receives a +0.1 bonus if its generated text contains certain predefined headers. This mechanism guides the model to produce more organized and predictable responses.

D. Representative Conversations

Example of Inter-Sample Feedback

Problem: Cybil and Ronda are sisters. The 10 letters from their names are placed on identical cards so that each of 10 cards contains one letter. Without replacement, two cards are selected at random from the 10 cards. What is the probability that one letter is from each sister's name? Express your answer as a common fraction. The answer is in the form $\frac{m}{n}$, where $\gcd(m, n) = 1$. Please provide the value of $m + n$.

Experience

Flow of thought

1. The problem is about probability, and the keywords "cards," "randomly selected," and "without replacement" indicate a combinatorial approach. We need the total number of ways to draw 2 cards out of 10 and the number of favorable ways where one card is from Cybil's name and one from Ronda's name.

2. Cybil's name has 5 letters, and Ronda's name has 5 letters. The total number of letters is 10, so the total number of ways to choose 2 cards is a combination of 10 taken 2 at a time:

$$\binom{10}{2} = 45.$$

3. For the favorable outcomes, choose 1 letter from Cybil's 5 letters and 1 letter from Ronda's 5 letters:

$$5 \times 5 = 25.$$

4. The probability is then the ratio of favorable outcomes to total outcomes:

$$\frac{25}{45} = \frac{5}{9}.$$

5. The final answer is the sum of the numerator and denominator of the simplified fraction:

$$5 + 9 = 14.$$

Takeaways

- Probability problems involving combinations can be solved by calculating total outcomes and favorable outcomes separately.
- The combination formula $\binom{n}{k}$ is useful for determining the number of ways to choose k items from n without replacement.
- Simplify fractions to their lowest terms by dividing the numerator and denominator by their greatest common divisor (gcd).
- Always ensure the answer is in the required format, such as a common fraction or the sum of numerator and denominator.

Exploration with Inter-Sample Feedback

Problem Statement

Let N be the number of 8-digit integers that can be formed using each of the digits $\{1, 2, 3, 4, 5, 6, 7, 8\}$ exactly once, such that the number is divisible by 22. Calculate the value of $N - 2025$.

Verification of the Solution Methodology

The core logic of the solution is broken down into several steps. Each step is verified below.

Step 1: Divisibility Rule for 22

A number is divisible by 22 if and only if it is divisible by both 2 and 11.

- **Divisibility by 2:** The last digit of the number must be even.
- **Divisibility by 11:** The difference between the sum of digits at odd positions and the sum of digits at even positions must be a multiple of 11.

Verification: Correct. This is the standard rule for divisibility by 22.

Step 2: Sum of Digits

The total sum of the available digits $\{1, 2, \ldots, 8\}$ is:

$$S = \sum_{i=1}^{8} i = \frac{8(8+1)}{2} = 36$$

Verification: Correct.

Step 3: Applying the Divisibility by 11 Rule

Let S_{odd} be the sum of the four digits in odd positions (D_1, D_3, D_5, D_7) and S_{even} be the sum of the four digits in even positions (D_2, D_4, D_6, D_8). We know:

$$S_{odd} + S_{even} = 36$$

For divisibility by 11, we need $S_{odd} - S_{even} = 11k$ for some integer k.

- If $k = 0$, then $S_{odd} = S_{even}$. This gives $2S_{odd} = 36 \implies S_{odd} = 18$. Thus, $S_{even} = 18$.
- If $k = 2$, then $S_{odd} - S_{even} = 22$. Combined with $S_{odd} + S_{even} = 36$, this gives $2S_{odd} = 58 \implies S_{odd} = 29$. This is impossible, as the maximum sum of four distinct digits from the set is $8 + 7 + 6 + 5 = 26$.
- Other non-zero values of k also lead to impossible sums.

Therefore, the only valid condition is $S_{odd} = S_{even} = 18$. **Verification:** Correct. This reasoning is sound.

Case-by-Case Analysis and Correction

We must satisfy two conditions simultaneously:

1. The last digit, D_8, must be even: $D_8 \in \{2, 4, 6, 8\}$.
2. The sum of digits in even positions, $S_{even} = D_2 + D_4 + D_6 + D_8$, must be 18.

We analyze each case based on the choice of D_8. For each case, we find the number of 3-digit combinations $\{a, b, c\}$ from the remaining digits that sum to $18 - D_8$. The number of ways to arrange these is 3! for the even positions and 4! for the remaining four digits in the odd positions.

Case 1: $D_8 = 2$

- Sum needed for $\{D_2, D_4, D_6\}$: $18 - 2 = 16$.
- Available digits for the sum: $\{1, 3, 4, 5, 6, 7, 8\}$.
- Combinations summing to 16: $\{1, 7, 8\}, \{3, 5, 8\}, \{3, 6, 7\}, \{4, 5, 7\}$.
- Total combinations: 4.
- Number of ways: $4 \times 3! \times 4! = 4 \times 6 \times 24 = 576$.

Case 2: $D_8 = 4$

- Sum needed for $\{D_2, D_4, D_6\}$: $18 - 4 = 14$.
- Available digits for the sum: $\{1, 2, 3, 5, 6, 7, 8\}$.
- Combinations summing to 14: $\{1, 5, 8\}, \{1, 6, 7\}, \{2, 5, 7\}, \{3, 5, 6\}$.
- Total combinations: 4.
- Number of ways: $4 \times 3! \times 4! = 4 \times 6 \times 24 = 576$.

Case 3: $D_8 = 6$ **(Correction)**

- Sum needed for $\{D_2, D_4, D_6\}$: $18 - 6 = 12$.
- Available digits for the sum: $\{1, 2, 3, 4, 5, 7, 8\}$.
- **Error Identification:** The original solution claimed 5 combinations. Let's re-evaluate systematically.
- Correct combinations summing to 12: $\{1, 3, 8\}, \{1, 4, 7\}, \{2, 3, 7\}, \{3, 4, 5\}$.
- Total combinations: 4.
- **Corrected number of ways:** $4 \times 3! \times 4! = 4 \times 6 \times 24 = 576$. The original calculation of 720 was **incorrect**.

Case 4: $D_8 = 8$

- Sum needed for $\{D_2, D_4, D_6\}$: $18 - 8 = 10$.
- Available digits for the sum: $\{1, 2, 3, 4, 5, 6, 7\}$.
- Combinations summing to 10: $\{1, 2, 7\}, \{1, 3, 6\}, \{1, 4, 5\}, \{2, 3, 5\}$.
- Total combinations: 4.
- Number of ways: $4 \times 3! \times 4! = 4 \times 6 \times 24 = 576$.

Final Calculation

Total Number of Valid Integers (N)

The total number of valid integers, N, is the sum of the ways from all four cases.

$$N = 576(\text{for } D_8 = 2) + 576(\text{for } D_8 = 4) + 576(\text{for } D_8 = 6) + 576(\text{for } D_8 = 8)$$

$$N = 4 \times 576 = 2304$$

Calculating the Final Difference

The problem asks for the value of $N - 2025$.

$$N - 2025 = 2304 - 2025 = 279$$

Conclusion

The verification process confirmed that the initial methodology was sound. However, a calculation error was found in the case where the last digit is 6. The original solution incorrectly identified 5 combinations instead of the correct 4. After correcting this error, the total number of valid integers is $N = 2304$. The final requested value is:

$$\boxed{279}$$

Exploration with Intra-Sample Feedback

Current Problem: Find the sum of all integer bases $b > 9$ for which 17_b is a divisor of 97_b.
Source Problem of the Feedback: Let the product $(12)(15)(16)$, with each factor written in base b, be equal to 3146 in base b. Let $s = 12 + 15 + 16$, with each term expressed in base b. Find the value of s in base b

Experience

Flow of thought

1. **Identify the Problem Type**: The problem involves numbers written in a variable base b. This immediately suggests that we need to convert these numbers into a common base (e.g., base 10) to manipulate them algebraically.
2. **Convert Numbers to Base 10**: For each number N_b (where N is represented in base b), express it in base 10. This allows us to handle the numbers using standard arithmetic.
3. **Set Up the Equation**: Write the given equation in terms of base 10 expressions. This step ensures that all terms are on the same footing and can be simplified algebraically.
4. **Expand and Simplify**: Expand any products or sums in the equation and simplify to form a polynomial equation in terms of b.
5. **Solve the Polynomial**: Find the roots of the polynomial equation. This step requires testing possible values or using algebraic methods to find the solution(s).
6. **Verify the Solution**: Plug the potential b values back into the original equation to ensure they satisfy the conditions.
7. **Calculate the Desired Value**: Once b is known, compute the sum s in base 10 and then convert it back to base b to get the final answer.

Takeaways

1. **Base Conversion**: Always convert numbers from variable bases to a standard base (e.g., base 10) for algebraic manipulation. It simplifies solving equations involving unknown bases.
2. **Expand Products**: When dealing with products in different bases, expand the terms fully to combine like terms and simplify the equation.
3. **Polynomial Roots**: Use polynomial roots to solve for unknowns in equations where variables represent bases. Rational Root Theorem is often useful for testing potential roots.
4. **Verification**: Always double-check solutions by substituting back into the original problem to ensure consistency.
5. **Algebraic Simplification**: Practice expanding and simplifying expressions to identify patterns and viable solutions.
6. **Number Representation**: Understand that the same number can be represented differently in different bases, and conversion formulas are essential for comparisons and calculations.

Response:

Experience Evaluation

1. **Analysis**: The provided experience involves a problem about converting numbers from a variable base to base 10, setting up equations, and solving for unknowns. This is relevant to the current problem because it also involves numbers in a variable base b and checking divisibility conditions.
2. **Decision**: While the experience provides a structured approach to solving problems with numbers in variable bases, the current problem is about evaluating integer bases for divisibility conditions. The experience's steps of converting to base 10, setting up equations, and solving for unknowns are directly applicable here.

3. **Justification**: The experience offers a systematic way to handle problems involving numbers in variable bases. It provides a clear flow of thought, from identifying the problem type to solving for unknowns. Therefore, I will modify the current problem-solving plan to align with this structured approach.

Final Plan

1. **Understand the Problem**: We are given two numbers in base b, 17_b and 97_b, and we need to find all integer bases $b > 9$ for which 17_b is a divisor of 97_b.
2. **Convert Numbers from Base b to Base 10**:
   - The number 17_b in base b can be expressed in base 10 as:
   $$17_b = 1 \cdot b + 7 = b + 7$$
   - The number 97_b in base b can be expressed in base 10 as:
   $$97_b = 9 \cdot b + 7$$
3. **Divisibility Condition**: We need 17_b to divide 97_b. In base 10 terms, this means:
   $$b + 7 \text{ divides } 9b + 7$$
   Mathematically, this can be written as:
   $$9b + 7 \equiv 0 \pmod{b + 7}$$
4. **Simplify the Divisibility Condition**: To check the divisibility condition $9b + 7 \equiv 0 \pmod{b + 7}$, we perform polynomial division or use modular arithmetic. We need to find b such that:
   $$9b + 7 \equiv 0 \pmod{b + 7}$$
   This means that $9b + 7$ should be divisible by $b + 7$. Perform polynomial division of $9b + 7$ by $b + 7$:
   - Divide the leading term $9b$ by the leading term b to get 9.
   - Multiply 9 by $b + 7$ to get $9b + 63$.
   - Subtract $9b + 63$ from $9b + 7$ to get:
   $$(9b + 7) - (9b + 63) = 7 - 63 = -56$$
   Therefore, we have:
   $$9b + 7 = (b + 7) \cdot 9 - 56$$
   So, the condition $9b + 7 \equiv 0 \pmod{b + 7}$ is equivalent to:
   $$-56 \equiv 0 \pmod{b + 7}$$
   This means $b + 7$ must be a divisor of -56, or equivalently, $b + 7$ must be a divisor of 56. The divisors of 56 are:
   $$\pm 1, \pm 2, \pm 4, \pm 7, \pm 8, \pm 14, \pm 28, \pm 56$$
5. **Find Valid b Values**: Since $b > 9$, we need $b + 7 > 16$, which means $b + 7$ must be a divisor of 56 that results in $b > 9$. Let's evaluate the divisors of 56:
   - The positive divisors of 56 are: $1, 2, 4, 7, 8, 14, 28, 56$.
   - We need $b + 7$ to be one of these, and solve for $b > 9$:
     - If $b + 7 = 14$, then $b = 14 - 7 = 7$ (which does not satisfy $b > 9$).
     - If $b + 7 = 28$, then $b = 28 - 7 = 21$ (which satisfies $b > 9$).
     - If $b + 7 = 56$, then $b = 56 - 7 = 49$ (which satisfies $b > 9$).

   So, the valid b values are 21 and 49.
6. **Sum the Valid b Values**: The integer bases $b > 9$ for which 17_b divides 97_b are 21 and 49. The sum of these bases is:
   $$21 + 49 = 70$$

Solution

1. **Convert Numbers**:
   - 17_b in base b is $1 \cdot b + 7 = b + 7$.
   - 97_b in base b is $9 \cdot b + 7$.
2. **Divisibility Condition**: We need $b + 7$ to divide $9b + 7$. This means:
$$9b + 7 \equiv 0 \pmod{b + 7}$$
3. **Find Divisors**: We need to find the values of b such that $b + 7$ divides $9b + 7$. We can express $9b + 7$ in terms of $b + 7$:
$$9b + 7 = 9(b + 7) - 56$$
   Therefore:
$$9b + 7 \equiv -56 \pmod{b + 7}$$
   For the remainder to be 0, $b+7$ must divide 56. The divisors of 56 are: $\pm 1, \pm 2, \pm 4, \pm 7, \pm 8, \pm 14, \pm 28, \pm 56$. Since $b > 9$, we have $b + 7 > 16$. So we only consider the divisors of 56 that are greater than 16:
   - $b + 7 = 28 \Rightarrow b = 21$
   - $b + 7 = 56 \Rightarrow b = 49$

   So, the possible values for b are 21 and 49.
4. **Sum the Valid Bases**: The integer bases $b > 9$ for which 17_b divides 97_b are 21 and 49. The sum of these bases is:
$$21 + 49 = 70$$

Now, let's verify the calculations:

- For $b = 21$:
  - $17_{21} = 1 \cdot 21 + 7 = 28$
  - $97_{21} = 9 \cdot 21 + 7 = 189 + 7 = 196$
  - We check if 28 divides 196:
$$196 \div 28 = 7$$
    Since 196 is divisible by 28, $b = 21$ works.
- For $b = 49$:
  - $17_{49} = 1 \cdot 49 + 7 = 56$
  - $97_{49} = 9 \cdot 49 + 7 = 441 + 7 = 448$
  - We check if 56 divides 448:
$$448 \div 56 = 8$$
    Since 448 is divisible by 56, $b = 49$ works.

Both $b = 21$ and $b = 49$ satisfy the conditions. Therefore, the sum of all such integer bases $b > 9$ is:

$$21 + 49 = 70$$

Final Answer

The sum of all integer bases $b > 9$ for which 17_b is a divisor of 97_b is:

$$\boxed{70}$$

E. Prompts

Zero-Shot Prompt

Let's think step by step and output the final answer within `\boxed{}`.

Summarizer System Prompt

You are an expert problem-solver who generates strategic thinking guides in a specific format.
Your task is to create a "Flow of Thought" guide based on the user's problem. This guide should be a reusable, first-person internal monologue that reveals an expert's strategic thinking process.
Core Principles:

1. **First-Person & Strategic:** Write the "Flow of thought" from an expert's perspective ("I," "my"). Don't just say *what* you're doing, explain *why* you're doing it. What clue in the problem triggered this step?
2. **General & Reusable:** Abstract the strategy. The "Flow of thought" should be a general blueprint for solving this *class* of problems, not just one specific instance.
3. **Synthesize Takeaways:** After detailing the thought process, distill the main strategies into a list of concise, high-level "Takeaways."

Output Format: Your response MUST strictly follow this structure:

ANALYSIS
(Your analysis of the user's request goes here.)

EXPERIENCE
Output Schema: The JSON object under `### Experience` must conform to this exact schema:

FLOW OF THOUGHT
A string containing the first-person internal monologue, formatted as a numbered list. Avoid specific problem details.",

TAKEAWAYS
A list of strings, where each string is a concise, high-level, and generalizable lesson not specific to the problem, which can be applied to other problems. Below is An Example of Experience

EXPERIENCE
FLOW OF THOUGHT
1. The request for the 'best' or 'shortest' route immediately signals a graph problem. My first step is to model it as such.
2. The cities become nodes, and the roads are edges. Since there are travel times, the edges are weighted.
3. With positive weights and a single destination, Dijkstra's algorithm is the ideal choice for finding the shortest path.

TAKEAWAYS
1. Keywords like 'best', 'shortest', or 'cheapest' often point to shortest path graph algorithms.

2. Always explicitly define your nodes, edges, and weights when modeling a problem as a graph.
3. For shortest path problems with non-negative weights, Dijkstra's is the standard algorithm.

Intra-Sample Feedback System Prompt

You are an AI expert in solution analysis. You will be given a problem and a previous experience for solving it. Your primary task is to perform a detailed, step-by-step analysis of the provided experience *before* reaching a final conclusion. You must show your reasoning process first and follow this exact structure:

1. STEP-BY-STEP VERIFICATION
   - **Goal:** Meticulously examine the logic and calculations of the provided experience.
   - **Action:** Break down the experience into its individual steps. For each step, explicitly state whether it is correct or not and briefly explain why.

Example for this section:

- **Step 1 (Calculation of X):** The experience calculated X as 5. **This is correct.** The formula A+B=C was applied properly with the given values.

2. CONCLUSION
   - **Goal:** State the final verdict based on your verification.
   - **Action:** Based *only* on the findings in your "Step-by-Step Verification," declare the overall experience as either **Correct** or **Incorrect**.

3. FINAL OUTPUT

Your response from this point forward depends on the verdict in your "Conclusion."

— **If the Conclusion was CORRECT, provide the following:**

VALIDATION

1. **Confirmation:** Reiterate that the previous experience and its solution are correct.
2. **Explanation:** Provide a holistic summary of why the solution is sound, referencing the key correct steps you identified in the verification phase.

— **If the Conclusion was INCORRECT, provide the following:**

CORRECTED SOLUTION

1. **Summary of Errors:** Briefly summarize the mistakes you identified during the "Step-by-Step Verification." Pinpoint exactly where the logic or calculations went wrong.
2. **New Step-by-Step Plan:** Propose and execute a new, correct plan. Clearly outline each step of your new approach.
3. **Final Answer:** Present the final, verified answer from your corrected plan.

Inter-Sample System Prompt

You are an expert problem-solver. You will be given a problem and previous experiences to guide your solution. Your task is to assess the experiences and then solve the problem. Your response MUST be organized into the

following three parts:

Experience Evaluation:

- Analyze the provided experiences.
- State whether you will follow some of them, modify them, or create a new plan entirely.
- Provide a clear justification for your decision. If you are modifying the plan, explain what changes you are making and why they are necessary for a more effective or accurate solution.

Final Plan:

- Outline the definitive, step-by-step plan that you will execute. This should be either the experience (if adopted) or your refined version.

Solution:

- Carry out your Final Plan step-by-step.
- Show all your work, calculations, and reasoning in detail.
- State the final answer clearly.

F. The Use of Large Language Models (LLMs)

In this work, LLMs are primarily employed for two purposes: (1) polishing the language of the manuscript to ensure grammatical correctness and coherence, and (2) assisting in the standardized organization and documentation of the released codebase. Importantly, all conceptual development, theoretical analysis, experimental design, and result interpretation are conducted independently by the authors. The use of LLMs is strictly limited to auxiliary tasks, ensuring that the scientific contributions of this paper remain entirely unaffected by such tools