# BOOTSTRAPPING LANGUAGE AND NUMERICAL FEEDBACK FOR REINFORCEMENT LEARNING IN LLMS

**Anonymous authors**
Paper under double-blind review

## 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 may seem intuitive, 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 as well as agentic abilities of large language models (LLMs) (OpenAI, 2024; Guo et al., 2025; Zeng et al., 2026; Team et al., 2026). 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., 2024a). 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).

Different from classic RL agents that simply map states to actions, LLMs can learn from explicit language feedback through context update. Existing studies have achieved notable progress in leveraging the ability to improve LLMs' reasoning performance without parameter change, such as self-reflection (Shinn et al., 2023), retrieving external information (Lewis et al., 2021), and reasoning templates (Yang et al., 2024b; Anthropic, 2025). We draw inspiration from such learning style, and advocate generating and utilizing language feedback during LLM RL to guide exploration and improve learning efficiency.

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, which we term as 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 (inter-sample feedback) often leads to *behavior collapse*, where the policy often under-utilizes retrieved context, generating near-identical

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

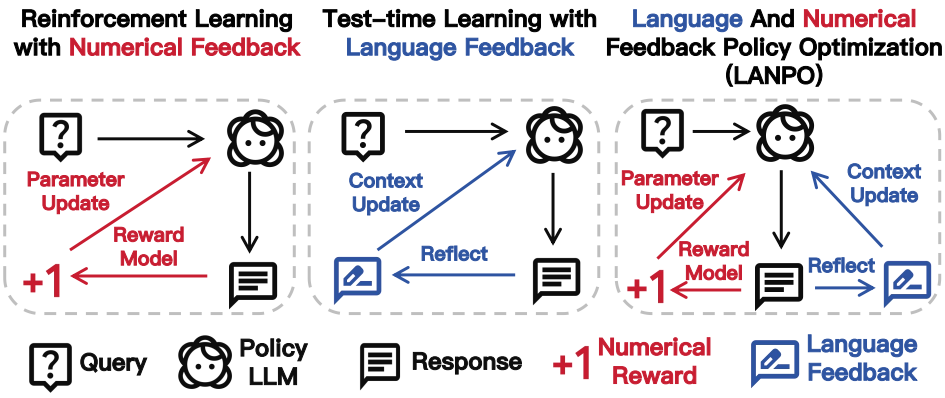



Figure 1: **Three paradigms for improving LLM reasoning.** Standard RL learns from scalar rewards, test-time methods adapt through language feedback, and LANPO combines language-guided exploration with reward-driven policy updates.

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)*, an RL training algorithm 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., 2024a). 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.

## 3 Challenges in Introducing Language Feedback




(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.

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.5, 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.3.

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

**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

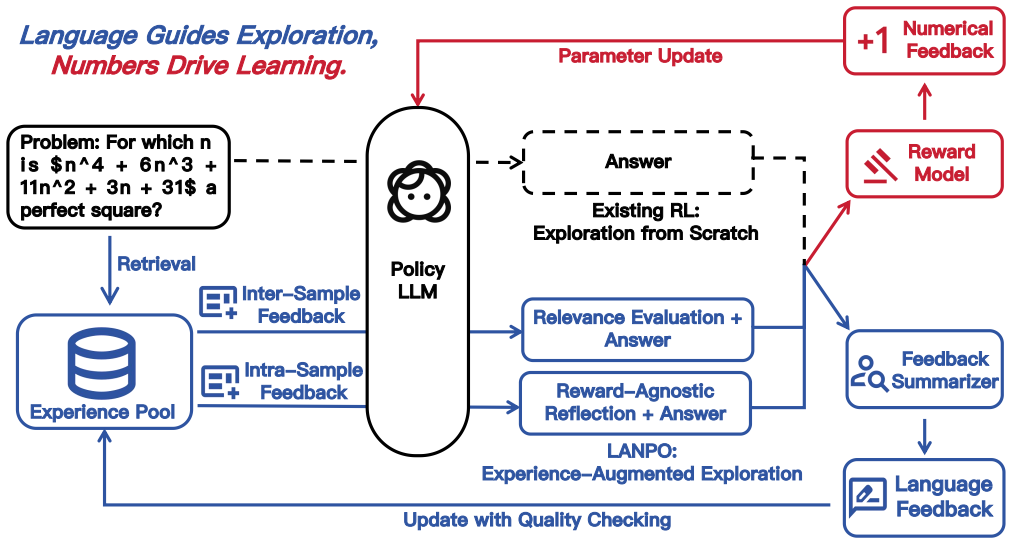



Figure 3: **LANPO training pipeline.** An evolving experience pool supplies related abstracted guidance and prior attempts for reflection, while scalar rewards evaluate all rollouts and drive policy-gradient updates.

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. Appendix A.1 provides a direct training-curve comparison of the two mechanisms (Figure 5), showing that both remain beneficial after the GRPO baseline begins to plateau.

### 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 their relevance to the problem at hand. Only experiences with a similarity score higher than a predefined threshold $\gamma$ are

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 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 $\tau_t$ 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. (2024b). 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 $\mathcal{D}$:

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

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. (2024b), and $\epsilon$ 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

Table 1: Math accuracy under different training and inference modes. AIME and AMC entries are mean ± standard deviation over 32 evaluation runs. ZS, SR, and Ret. denote zero-shot, self-reflection, and retrieval; bold marks the best value for each model and benchmark.

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

Table 2: Coding scores for Qwen2.5-7B-Instruct-SFT. Entries are mean ± standard deviation over 32 evaluation runs; abbreviations follow Table 1.

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

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.

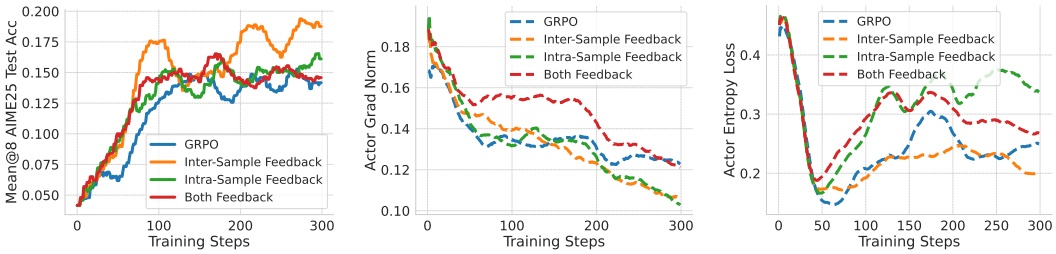
**Models and datasets.** We evaluate three base models: Qwen3-4B, Qwen2.5-7B and Qwen3-14B (Yang et al., 2024a; 2025a). For math RL, we adopt the DAPO dataset (Yu et al., 2025) containing 17K competition-level problems. Performance is measured on AIME25, AIME24, AMC23, and MATH-500 (Hendrycks et al., 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., 2024b) 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 $\gamma$ to be 0.9, Complete details are provided 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




(a) AIME25 accuracy. (b) Gradient norm. (c) Action entropy.

Figure 4: **Training dynamics on Qwen2.5-7B.** LANPO improves **(a)** AIME25 accuracy while retaining stable **(b)** gradient norms. **(c)** Reflection sustains higher entropy, whereas retrieved abstractions focus exploration.

Table 3: **Relevance-filtering ablation.** Filtering ($\gamma = 0.9$) makes retrieval consistently useful and gives the best retrieval-augmented average.

| Training | Inference | AIME 25 | AIME 24 | AMC | MATH | Avg. |
|---|---|---|---|---|---|---|
| GRPO | ZS | 13.85 | 17.81 | 58.51 | 81.00 | 42.79 |
| | +Ret. | 10.73 | 13.33 | 61.78 | 72.40 | 39.56 |
| LANPO, $\gamma = 0$ | ZS | 15.62 | **20.31** | 65.14 | 81.20 | 45.57 |
| | +Ret. | 15.94 | 17.29 | **70.60** | 81.20 | 46.26 |
| LANPO, $\gamma = 0.9$ | ZS | 17.19 | 18.23 | 60.50 | **82.80** | 44.68 |
| | +Ret. | **17.81** | 20.00 | 69.28 | 82.40 | **47.37** |

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 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.8 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 the AIME25 benchmark, as illustrated in Figure 4. Both intra- and inter-sample feedback mechanisms clearly outperform the GRPO baseline, achieving faster convergence and a higher final test accuracy (Figure 4a). These performance gains are realized without compromising stability, as evidenced by the smooth decline in the actor gradient norm for all methods (Figure 4b).

The plot of action entropy in Figure 4c 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 low-

est 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.7.

**Ablation: Necessity of Filtering.** We ablate the *training-time* relevance filtering used in inter-sample feedback by setting the similarity threshold $\gamma$ 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 $\Delta_{\text{NED}}$ between zero-shot and retrieval-conditioned outputs (Table 5, Appendix A.3). Filtering generally increases $\Delta_{\text{NED}}$, suggesting reduced invariance to the injected experience and improved ability to utilize inter-sample feedback.

**Ablation: Effect of Feedback Ratio.** Appendix A.2 studies the probability $p$ of including feedback during LANPO training, with the complete benchmark results reported in Table 4. We adopt intra-sample feedback for this ablation. A low value ($p = 0.25$) underuses past experience, while $p = 1.0$ produces the lowest zero-shot average (42.13), consistent with the absence of from-scratch rollouts during training. The best trade-off occurs at $p = 0.75$, which reaches the highest average accuracy with self-reflection (45.80). This confirms that balancing feedback-aware and from-scratch rollouts is important for 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.9. 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.

### AI USE STATEMENT

In this work, we used generative AI tools for polish writing, all of which we have reviewed. We have not used generative AI tools for proposing ideas, and AI designing research methodology is not applicable to this work. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

### REPRODUCIBILITY STATEMENT

We have provided both the high-level and low-level details of our method, including all hyperparameters that we use in Appendix B. Our training and evaluations are performed on open-source LLMs and benchmarks, with all specific prompts used in Appendix E. Our RL algorithms and infrastructure extends the public implementation of VeRL (Sheng et al., 2024). We will open-source the necessary code to implement our ideas upon acceptance, with which we believe the research community will be able to replicate our findings.

## REFERENCES


Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*, 2025.

Zachary Ankner, Bai Liu, Ming Sun, et al. Critique-out-loud reward models. *arXiv preprint arXiv:2408.11791*, 2024.

Anthropic. Agent skills. `https://platform.claude.com/docs/en/agents-and-tools/agent-skills/overview`, 2025. Accessed: 2026-09-21.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

Carlo Baronio, Pietro Marsella, Ben Pan, Simon Guo, and Silas Alberti. Kevin: Multi-turn rl for generating cuda kernels. 2025.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *NeurIPS*, 33:1877–1901, 2020.

Angelica Chen, Jérémy Scheurer, Jon Ander Campos, Tomasz Korbak, Jun Shern Chan, Samuel R. Bowman, Kyunghyun Cho, and Ethan Perez. Learning from natural language feedback. *Transactions on Machine Learning Research*, 2024.

Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu, Qixin Xu, Weize Chen, et al. Process reinforcement through implicit rewards. *arXiv preprint arXiv:2502.01456*, 2025.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei

An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. Deepseek-v3 technical report. 2025.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, et al. A survey on llm-as-a-judge. *The Innovation*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Jingkai He, Tianjian Li, Erhu Feng, Dong Du, Qian Liu, Tao Liu, Yubin Xia, and Haibo Chen. History rhymes: Accelerating llm reinforcement learning with rhymerl. 2025.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei M Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal Behbahani, and Aleksandra Faust. Training language models to self-correct via reinforcement learning. 2024.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. 2021.

Xiaoya Li, Xiaofei Sun, Jiwei Li, Albert Wang, and Chris Shum. Cuda-l1: Improving cuda optimization via contrastive reinforcement learning. 2025.

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

Aman Madaan, Niket Tandon, Prakhar Gupta, 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/`.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L Wainwright, et al. Training language models to follow instructions with human feedback. In *NeurIPS*, 2022.

Yuxiao Qu, Amrith Setlur, Virginia Smith, Ruslan Salakhutdinov, and Aviral Kumar. Pope: Learning to reason on hard problems via privileged on-policy exploration, 2026.

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

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

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

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

Zeyu Shao, Peng Wang, Qinkai Zhu, Rui Xu, Jiaxing Song, Ming Zhang, Yuxiao Li, Yuhuai Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024a.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, YK Li, Y Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024b.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv:2409.19256*, 2024.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y Charles, et al. Kimi k3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

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

Jun Wang, Zihan Wang, Linyi Yang, Huichi Zhou, Kun Shao, Siyuan Guo, Yihang Chen, Guchun Zhang, Xue Yan, Kin Hei Lee, and Ka Yiu Lee. Agentfly: Fine-tuning llm agents without fine-tuning llms. 2025.

Yifei Wang, Yuyang Wu, Zeming Wei, Stefanie Jegelka, and Yisen Wang. A theoretical understanding of self-correction through in-context alignment. In *NeurIPS 2024*, 2024a.

Yifei Wang, Yuyang Wu, Zeming Wei, Stefanie Jegelka, and Yisen Wang. A theoretical understanding of self-correction through in-context alignment. *NeurIPS 2024*, 2024b.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024a.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. 2025a.

Ling Yang, Zhaochen Yu, Tianjun Zhang, Shiyi Cao, Minkai Xu, Wentao Zhang, Joseph E. Gonzalez, and Bin Cui. Buffer of thoughts: Thought-augmented reasoning with large language models. 2024b.

Ling Yang, Zhaochen Yu, Bin Cui, and Mengdi Wang. Reasonflux: Hierarchical llm reasoning via scaling thought templates. 2025b.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. Self-rewarding language models. *arXiv preprint arXiv:2401.10020*, 2024.

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

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *arXiv preprint arXiv:2306.05685*, 2023.

Daniel M Ziegler, Nisan Stiennon, Jeffrey Wu, Tom Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

# A Additional Results

## A.1 Effectiveness of the Feedback Designs

We first isolate the two mechanisms that make language feedback safe to use during RL training. Figure 5 compares the AIME24 test accuracy of standard GRPO with inter-sample feedback using relevant abstraction and intra-sample feedback using reward-agnostic reflection. This comparison complements the final-checkpoint results in the main paper by showing how the benefit develops over the course of training.

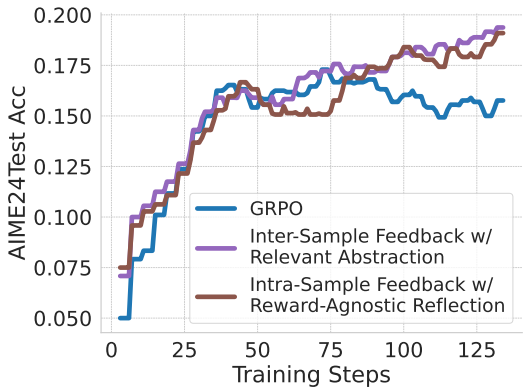



Figure 5: **Training-time effectiveness of LANPO's feedback designs on AIME24.** Relevant abstraction makes inter-sample feedback useful, while reward-agnostic reflection enables intra-sample improvement without exposing gold solutions.

Both LANPO variants initially improve at a rate similar to GRPO, but their trajectories separate later in training. GRPO peaks earlier and then settles near a lower accuracy, whereas both feedback-guided variants continue to improve and finish around $0.19$ accuracy. The sustained gains indicate that the language feedback is not merely accelerating the initial warm-up: it continues to provide useful exploration guidance after the baseline begins to plateau. The inter-sample variant finishes slightly higher, while the intra-sample variant follows a comparable trend. Together, these results support the two design choices from complementary directions: relevant abstraction makes retrieved experience actionable, and reward-agnostic reflection extracts useful information from prior attempts without relying on leaked labels.

## A.2 Sensitivity to the Feedback Ratio

We next study the training-time feedback ratio $p$, i.e., the probability that a rollout is conditioned on language feedback rather than generated from scratch. Table 4 reports both zero-shot (ZS) inference and inference with self-reflection (+SR) for each ratio, with all other training settings held fixed.

Table 4: Influence of the feedback ratio $p$. A moderate ratio ($p = 0.75$) gives the best average accuracy by balancing feedback-aware and from-scratch rollouts.

| $p$ | Inference | AIME 25 | AIME 24 | AMC | MATH | Avg. |
|---|---|---|---|---|---|---|
| 0.00 | ZS | 13.85 | 17.81 | 58.51 | 81.00 | 42.79 |
| | +SR | 15.62 | 21.56 | 58.58 | 80.60 | 44.09 |
| 0.25 | ZS | 15.62 | 18.96 | 57.79 | 79.20 | 42.89 |
| | +SR | **18.02** | 19.90 | 58.47 | 82.40 | 44.70 |
| 0.50 | ZS | 12.60 | 20.00 | 62.05 | 79.80 | 43.61 |
| | +SR | 14.48 | 21.04 | **62.50** | 80.60 | 44.66 |
| 0.75 | ZS | 15.62 | 20.94 | 58.51 | 82.00 | 44.27 |
| | +SR | 17.40 | **23.54** | 58.85 | **83.40** | **45.80** |
| 1.00 | ZS | 14.06 | 16.35 | 56.29 | 81.80 | 42.13 |
| | +SR | 16.04 | 21.15 | 60.24 | 82.20 | 44.91 |

The effect of $p$ is non-monotonic. Increasing the ratio from $0$ to $0.75$ steadily improves the zero-shot average from $42.79$ to $44.27$, and $p = 0.75$ also achieves the best self-reflection average of $45.80$. In contrast, using feedback for every rollout ($p = 1.0$) lowers zero-shot performance to $42.13$, suggesting that the policy becomes overly dependent on feedback-conditioned contexts and receives too little practice solving problems independently. Smaller ratios underuse the accumulated experience, although individual benchmarks sometimes favor them (e.g., AIME25 at $p = 0.25$ and AMC at $p = 0.50$). Overall, $p = 0.75$ provides the most reliable cross-benchmark trade-off: it exploits past experience frequently while preserving enough from-scratch rollouts to maintain robust zero-shot reasoning.

### A.3 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 $\varepsilon$. Intuitively, a large $p_{\text{collapse}}$ means the policy becomes nearly invariant to the presence of experience.

In our experiments, we instantiate $\Delta$ 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 $\Delta_{\text{NED}}$ indicates that $c$ has weak behavioral effect (collapse).

**An example of behavior collapse.** We provide a representative example in Example A.3. 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.

**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.

**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 $\Delta_{\mathrm{NED}}$ between experience-conditioned responses and their zero-shot counterparts. Enabling filtering consistently increases $\Delta_{\mathrm{NED}}$, 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 $\Delta_{\mathrm{NED}}$ 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) |

### A.4 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.5 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.

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

The results show that leaking partial reference solutions does not lead to consistent improvement over the no-leakage 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.6 More Evaluation Results

Table 7 presents the evaluation of SFT and GRPO models when language feedback is provided during inference. Two key findings emerge from the results. First, for models trained via Reinforcement

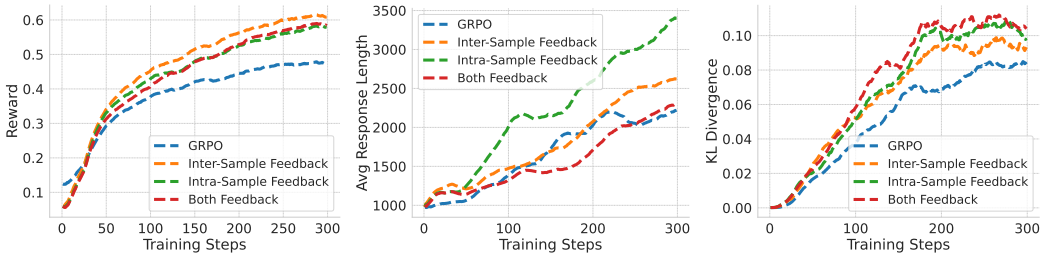
Table 7: Testing SFT & GRPO trained models with language feedback.

| 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 |

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.7 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.8 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.9 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

Table 8: Performance comparison between GRPO and LANPO on math benchmarks. The base model here is the Qwen3-4B-Instruct 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-Instruct -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 |

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** |

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

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 | 13.85 | 17.81 | ˜72 hours | ˜0.04M | ˜0.5M |
| GRPO-730 steps | 15.90 | 18.60 | ˜160 hours | ˜0.08M | ˜1M |
| LANPO-Both-330 steps | **16.25** | **20.62** | ˜146 hours | ˜0.2M | ˜0.6M |

(∼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.

# 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 $\gamma$ 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 decay function in the form of $w_i \propto 1/(i+1)$, 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 Intra-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 Inter-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.