THINKTWICE: JOINTLY OPTIMIZING LARGE LANGUAGE MODELS FOR REASONING AND SELF-REFINEMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning with verifiable rewards (RLVR) has advanced the reasoning capabilities of large language models (LLMs) to the point where frontier models match or exceed human experts on many tasks. Self-refinement, where a model reviews and revises its own solutions, can recover problems that its first attempt narrowly misses. However, prompt-only refinement remains brittle even for frontier models, and existing training-based methods rely on external guidance such as critique annotations or explicit signals indicating whether the initial answer is correct. In this work, we introduce ThinkTwice, a simple two-phase RLVR framework that jointly optimizes an LLM to solve reasoning problems and refine its own answers, without any external signals. Concretely, ThinkTwice replaces every even GRPO step with a refinement step, which optimizes the model to revise its own solutions from the preceding step under the same binary correctness reward, so training takes only 3% longer than GRPO. Across five mathematical reasoning benchmarks and two model families including Qwen3-4B and Olmo3-7B, ThinkTwice improves both reasoning and refinement performance over competitive online policy optimization baselines. Specifically, on Qwen3-4B, ThinkTwice outperforms GRPO on AIME by 5 percentage points before refinement and by 11.5 points after one self-refinement step. Analysis of the training dynamics of ThinkTwice reveals an implicit *rectify-then-fortify* curriculum: refinement predominantly corrects errors early in training and naturally shifts toward preserving already-correct solutions as the model improves, yielding a more rectified reward signal. Our work establishes joint training of reasoning and self-refinement as a principled and effective methodology for RLVR.[1]

1 INTRODUCTION

Reinforcement learning with verifiable rewards (RLVR) has emerged as an effective paradigm for improving the reasoning capabilities of large language models (LLMs) (Shao et al., 2024; Guo et al., 2025). Yet even strong reasoners can produce solutions that contain correctable errors, such as incomplete derivations, algebraic mistakes, or unproductive solution paths. A natural strategy from human problem-solving is *self-refinement*: people routinely revisit initial solutions, identify errors, and revise their reasoning (Polya, 1945). There is growing evidence that self-refinement can similarly help LLMs recover on challenging problems where initial attempts narrowly fail (Gou et al., 2023; Weng et al., 2023).

However, existing approaches to improving self-refinement in LLMs, broadly categorized into training-free and training-based methods, each have notable limitations. First, training-free methods (Madaan et al., 2023; Shinn et al., 2023) prompt the model to critique and revise at inference time but do not learn a reusable refinement policy. Moreover, prompt-only refinement remains brittle even for frontier models. As illustrated in Figure 1(A), we observe a slight performance *decrease* from the frontier LLM on AIME24 after self-refinement prompting.

Second, although training-based methods learn refinement behavior, existing methodologies typically rely on process supervision (Zhang et al., 2024a; Cui et al., 2025), critique annotations (Zhang et al.,

[1]The codebase is available at https://anonymous.4open.science/r/ThinkTwice-2253.

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.

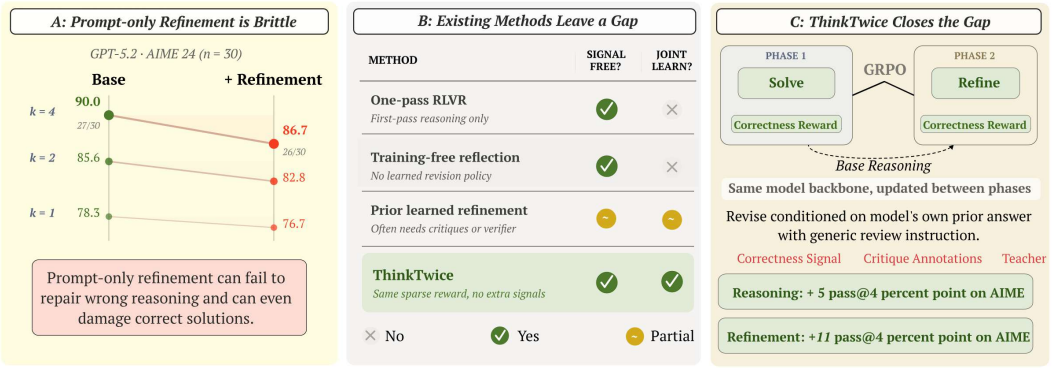

Figure 1: (A) Prompt-only reflection can reduce top frontier LLM's performance on AIME24, indicating brittleness. (B) ThinkTwice compared with existing method families. (C) ThinkTwice addresses these gaps by sequentially training a shared model backbone—first solving, then refining—yielding significant gains (+5 points reasoning, +11 points refinement) on AIME with Qwen3-4B.

2025b), or explicit signals indicating whether the initial answer is correct (Ma et al., 2025). Such supervision is not always available in practice: at the frontier, no stronger model exists to provide reliable critique, and human oversight may be insufficient for challenging problems (Burns et al., 2023).

In this work, we introduce ThinkTwice, a unified RLVR framework that jointly optimizes reasoning and self-refinement, in which the model refines entirely on its own: nothing tells it whether its previous attempt was correct or where it went wrong. In each pair of training steps, ThinkTwice first optimizes the model on solving a batch of reasoning problems, then optimizes it on refining its own solutions to the same problems ("thinking twice"), using a generic review instruction and the same binary correctness reward in both phases. Each phase is one training step and samples as many solutions as a GRPO step. ThinkTwice therefore uses half as many training problems per step as GRPO. The refinement phase gives the model a second attempt at each problem conditioned on its prior solution, allowing it to recover from correctable errors that the first attempt missed. As shown in Figure 1(B, C), ThinkTwice serves as a simple, effective method that doesn't require correctness signals, critique annotations, or external teacher models.

We train ThinkTwice with Qwen3-4B and Olmo3-7B and evaluate on a suite of five mathematical reasoning benchmarks. Each model first answers every problem directly and then refines its own solutions, and all methods use the same number of samples and the same fixed token budget in both stages. ThinkTwice substantially improves both reasoning and self-refinement capabilities over competitive baselines. Specifically, on Qwen3-4B, ThinkTwice outperforms GRPO on the challenging AIME problems by 5 percentage points before refinement and by 11.5 points after one step of self-refinement, measured by pass@4. Across both models, ThinkTwice achieves the highest average performance for both reasoning and self-refinement, and when used as a refiner applied to other baselines' solutions, ThinkTwice also achieves the highest score. At an equal number of generations, refinement raises ThinkTwice's pass@k more than additional sampling does (Appendix C.2).

To understand how ThinkTwice works, we analyze its training dynamics and find that the refinement phase gives rise to an implicit *rectify-then-fortify* curriculum: early in training, refinement predominantly corrects errors in failed solutions, and as the model improves, it naturally shifts toward preserving and polishing already-correct ones. This dynamic yields a more rectified reward signal, as the refinement phase recovers useful learning signal from problems where the base attempt alone falls short. Additionally, ThinkTwice adds only 3% to total wall-clock time at matched update counts, and in our runs it reached its best checkpoint in 16% less wall-clock time than GRPO.

2 Related Work

Reinforcement learning with verifiable rewards (RLVR). Recent progress in reasoning post-training has been driven by reinforcement learning with verifiable rewards (RLVR), where models

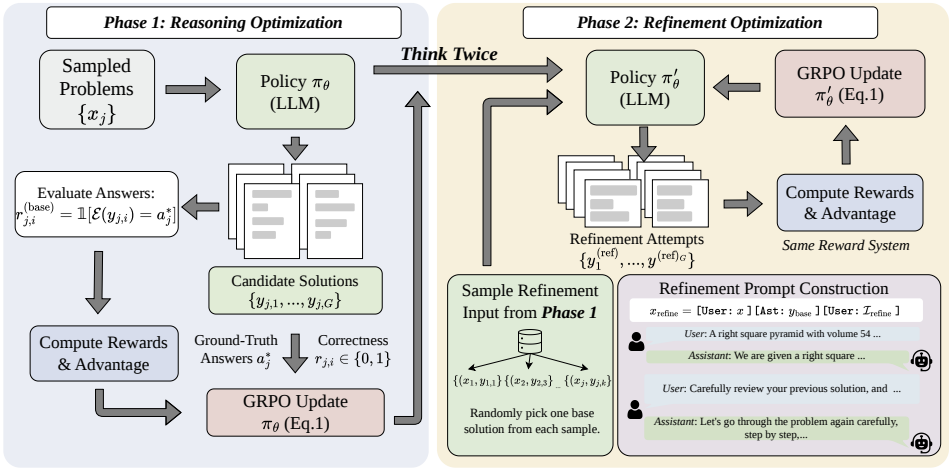

Figure 2: ThinkTwice at a glance.

are optimized using automatically checkable outcome signals such as exact answers. DeepSeekMath introduced Group Relative Policy Optimization (GRPO), a practical critic-free PPO-style recipe for this setting, and DeepSeek-R1 further showed that large-scale outcome-only RL can elicit strong reasoning behavior without process supervision (Shao et al., 2024; Guo et al., 2025). A large follow-up literature has since expanded the RLVR design space. Open reproductions and training recipes such as Open-Reasoner-Zero, SimpleRL-Zoo, and Skywork-OR1 helped make large-scale RLVR more reproducible (Hu et al., 2025; Zeng et al., 2025b; He et al., 2025; Fu et al., 2025). On the optimization side, DAPO improves long-CoT training stability; Dr. GRPO analyzes optimization bias in GRPO; GSPO moves from token-level to sequence-level importance ratios and clipping; and newer variants such as GMPO, GPG, and shrinkage baselines revisit ratio aggregation, simplification, response-length bias, and baseline variance (Yu et al., 2025a; Liu et al., 2025; Zheng et al., 2025; Zhao et al., 2025b; Chu et al., 2025; Zeng et al., 2025a).

Self-refinement. Self-refinement aims to improve an initial response by generating feedback or verification and then revising the answer (Madaan et al., 2023). A first class of methods is training-free and operates purely at inference time, such as Self-Refine, Reflexion, self-verification prompting, CRITIC, and self-consistency decoding (Madaan et al., 2023; Shinn et al., 2023; Weng et al., 2023; Gou et al., 2023; Wang et al., 2022). These approaches show that extra test-time computation, self-feedback, or tool-based critique can improve outputs, but they do not learn a reusable refinement policy; moreover, prompt-only self-correction can be unreliable for reasoning without external feedback (Huang et al., 2023). A second class of work trains refinement behavior more directly, including methods based on verifiers (Cobbe et al., 2021; Zhang et al., 2024b), process supervision (Uesato et al., 2022; Lightman et al., 2023; Wang et al., 2024; Yuan et al., 2024; Cui et al., 2025), critique data (Xi et al., 2024; Yu et al., 2025b), synthetic correction traces (Welleck et al., 2022; Qu et al., 2024; Xiong et al., 2025; Zhao et al., 2025a), or multi-turn RL with explicit self-verification or critique objectives (Ma et al., 2025; Jiang et al., 2025; Zhang et al., 2025a;b) or a reward that scores the first attempt (Kumar et al., 2024). ThinkTwice is closest to the RL-based self-refinement line, but is deliberately simpler: it uses a shared policy, a generic review instruction, and the same final-answer correctness reward in both phases, without process labels, critique annotations, correctness hints, or a reward that depends on whether the initial answer is correct. A more detailed comparison is provided in Appendix A.

3 Methodology

We introduce ThinkTwice, an RLVR framework that jointly optimizes reasoning and self-refinement without verification, i.e., without access to external signals indicating whether the intermediate steps or solutions are correct or not. We briefly recap GRPO in Section 3.1, then describe how we formulate the self-refinement problem in Section 3.2, and finally explain how we integrate both reasoning and self-refinement optimization in Section 3.3.

3.1 PRELIMINARIES

We use Group Relative Policy Optimization (GRPO) (Shao et al., 2024; Guo et al., 2025) as our backbone reinforcement learning (RL) algorithm. GRPO eliminates the need for a separate critic model by computing advantages from grouped samples, which offers better stability and efficiency during LLM RL.

Let π_θ denote the language model policy parameterized by θ. For a given input x, GRPO samples a group of G responses $\{y_1, \ldots, y_G\} \sim \pi_{\theta_{\text{old}}}(\cdot|x)$, where each generation of y_i is often called a rollout, and optimizes:

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{x,\{y_i\}_{i=1}^{G}} \left[\frac{1}{G} \sum_{i=1}^{G} \frac{1}{|y_i|} \sum_{t=1}^{|y_i|} \mathcal{L}_{i,t} - \beta \mathbb{D}_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}) \right], \tag{1}$$

where $\mathcal{L}_{i,t} = \min\left(\rho_{i,t} A_i, \text{clip}(\rho_{i,t}, 1-\epsilon, 1+\epsilon) A_i\right)$ denotes the clipped surrogate objective, $\rho_{i,t} = \pi_\theta(y_{i,t}|x, y_{i,<t}) / \pi_{\theta_{\text{old}}}(y_{i,t}|x, y_{i,<t})$ is the importance ratio of the t-th token of y_i, ϵ is the clipping threshold, and β controls the KL penalty against the reference policy π_{ref}.

The advantage A_i is computed via group normalization:

$$A_i = \frac{r_i - \text{mean}(r_1, \ldots, r_G)}{\text{std}(r_1, \ldots, r_G)}, \tag{2}$$

where r_i is the reward for response y_i. For mathematical reasoning tasks with verifiable answers, we use outcome-based correctness rewards: $r_i = \mathbb{1}[\mathcal{E}(y_i) = a^*]$, where $\mathcal{E}$ extracts the answer from y_i and a^* is the ground-truth answer.

3.2 SELF-REFINEMENT FORMULATION

In our self-refinement framework, we use *base solution* to refer to an initial solution attempt that the model will subsequently self-refine into a *refined solution*. Given a problem x and a base solution y_{base} generated by the current policy, we construct the refinement prompt as a multi-turn conversation:

$$x_{\text{refine}} = \texttt{[User: } x \texttt{][Ast: } y_{\text{base}} \texttt{][User: } \mathcal{I}_{\text{refine}} \texttt{]}, \tag{3}$$

where $\mathcal{I}_{\text{refine}}$ is a task-agnostic instruction that asks the model to review and improve its previous solution. Crucially, $\mathcal{I}_{\text{refine}}$ contains no indication of whether y_{base} is correct or incorrect, distinguishing our approach from methods that rely on external critique signals. We use a general refinement instruction for ThinkTwice as shown in Appendix B.1.

3.3 THINKTWICE

ThinkTwice alternates between two training phases within a unified GRPO framework, as depicted in Figure 2. In each pair of training steps, the model first optimizes on reasoning for a batch of problems, then optimizes on refining its own solutions to the same problems. This exposes the model to each problem twice per training batch under complementary objectives: solving from scratch and improving upon its own prior attempt.

Phase 1: Reasoning Optimization. For each problem x sampled from the training set $\mathcal{D}$, we generate G candidate solutions $\{y_1, \ldots, y_G\}$ using the current policy π_θ. Each solution is evaluated with the correctness reward $r_i^{(\text{base})} = \mathbb{1}[\mathcal{E}(y_i) = a^*]$, and the policy is updated via Eq. (1), yielding π'_θ. From the G rollouts per problem x, we randomly select one base solution for the subsequent refinement phase.

Phase 2: Refinement Optimization. Using the selected base solutions from Phase 1, we construct refinement prompts following Eq. (3). The model with updated policy π'_θ then generates G refinement attempts $\{y_1^{(\text{ref})}, \ldots, y_G^{(\text{ref})}\}$ for each refinement prompt. Since π'_θ has already been updated using reward signal from x during Phase 1, it is better equipped to improve upon the base solutions produced

Algorithm 1 ThinkTwice: Jointly Optimizing Reasoning and Refinement

Require: Training dataset $\mathcal{D}$, current policy π_θ, training batch size B, group size G, refinement instruction $\mathcal{I}_{\text{refine}}$

for each training iteration $t = 1, 2, \ldots, T$ **do**
  // **Phase 1: Reasoning**
  Sample batch of problems $\{x_j\}_{j=1}^{B}$ from $\mathcal{D}$
  for each problem x_j **do**
    Generate solutions $\{y_{j,1}, \ldots, y_{j,G}\} \sim \pi_\theta(\cdot|x_j)$
    Compute rewards $r_{j,i} = \mathbb{1}[\mathcal{E}(y_{j,i}) = a_j^*]$ for $i = 1, \ldots, G$
  end for
  Update π_θ to π'_θ on reasoning samples
  Sample refinement pairs from: $\{(x_j, y_{j,i})\}_{j,i}$
  // **Phase 2: Refinement**
  for each (x, y_{base}) in sampled pairs **do**
    Construct x_{refine} via Eq. (3)
    Generate refinements $\{y_1^{(\text{ref})}, \ldots, y_G^{(\text{ref})}\} \sim \pi'_\theta(\cdot|x_{\text{refine}})$
    Compute rewards $r_i^{(\text{ref})} = \mathbb{1}[\mathcal{E}(y_i^{(\text{ref})}) = a^*]$ for $i = 1, \ldots, G$
  end for
  Update π'_θ to π''_θ on refinement samples
end for

by π_θ. Each refined solution is evaluated using the same correctness reward $r_i^{(\text{ref})} = \mathbb{1}[\mathcal{E}(y_i^{(\text{ref})}) = a^*]$, and the policy is updated via Eq. (1) as well, yielding π''_θ.

We now elaborate on two key design choices in the refinement optimization phase.

Base Solution Sampling. Since GRPO relies on multiple rollouts for advantage estimation, we accordingly have multiple base solutions as candidates for y_{base}. In ThinkTwice, we adopt the simplest sampling strategy: randomly picking one base solution per problem x. This design not only inherently covers the full spectrum of training samples, but also aims to establish an *emergent curriculum*: early in training, when the model frequently produces incorrect solutions, the refinement phase predominantly trains on error correction. As reasoning accuracy improves, fewer incorrect solutions are available, and the refinement phase naturally shifts toward polishing already-correct solutions. This adaptive curriculum naturally steers training toward appropriately challenging refinement scenarios aligned with the model's current capability boundaries.

Refined Solution Reward. While $\mathcal{I}_{\text{refine}}$ provides high-level guidance for the refinement task, we do not use extra reward signals to enforce adherence to its prescribed structure—through RL, the model is free to develop its own refinement strategies. ThinkTwice solely employs the binary signal that depends on the correctness of the refined solution. This reward signal encourages the model to: **(1)** detect and correct errors when the base solution is wrong, or **(2)** preserve and polish when the base solution is already correct. Because the refinement is formulated as a standard multi-turn conversation, the optimization process is seamlessly handled by native GRPO.

The training procedure is detailed in Algorithm 1 and Appendix B.1.

4 Results

4.1 Experimental Setup

We follow the training and evaluation protocol of Dr. GRPO (Liu et al., 2025). We train on math questions from the MATH training dataset (Hendrycks et al., 2021) and evaluate on a suite of five benchmarks: AIME, AMC, MATH500, Minerva Math (Lewkowycz et al., 2022), and OlympiadBench (He et al., 2024). All methods use the binary correctness reward from a boxed answer grader and Math-Verify,[2] which check the final answer against the ground truth.

[2] https://github.com/huggingface/Math-Verify

Table 1: Reasoning performance (pass@4, ↑) across five mathematical reasoning benchmarks. **Bold** and underline denote the best and second-best results per dataset within each model.

Method	AIME	AMC	MATH500	Minerva	OlympiadBench	Average
Qwen3-4B						
Base Model	29.18	64.87	88.47	39.61	57.90	56.01
GRPO	39.06	75.36	91.86	41.03	63.80	62.22
DrGRPO	35.46	77.65	91.83	42.75	66.51	62.84
DAPO	41.15	79.10	**93.67**	42.68	**68.39**	65.00
ThinkTwice	**44.11**	**79.59**	93.60	**42.94**	67.60	**65.57**
OLMo3-7B						
Base Model	32.81	68.77	89.87	40.63	61.36	58.69
GRPO	**39.38**	77.05	92.28	41.13	62.42	62.45
DrGRPO	36.09	74.33	91.65	42.07	65.09	61.85
DAPO	36.72	76.16	91.56	42.39	65.71	62.51
ThinkTwice	39.24	**79.89**	**92.74**	**43.43**	**65.81**	**64.22**

We experiment with two instruction-tuned models, Qwen3-4B-Instruct-2507 (Yang et al., 2025) and OLMo3-7B-Instruct (Olmo et al., 2025), as the refinement phase requires the ability to follow multi-turn instructions. Our baselines include standard GRPO (Shao et al., 2024), Dr. GRPO (Liu et al., 2025), and DAPO (Yu et al., 2025a). We also include two training-free baselines, Reflexion (Shinn et al., 2023) and Self-Refine (Madaan et al., 2023), for refinement evaluation. Our experiments are implemented using VERL (Sheng et al., 2024),[3] with implementation details and hyperparameter configurations provided in Appendix B.1.

A step is one policy update on 32 prompts with 8 rollouts each, and step counts, including the checkpoint steps in Figures Figure 4 and Figure 5, include both ThinkTwice phases; Table 3 details each method's compute. All methods are compared at matched numbers of policy updates, rollouts per update, and generation budgets. Because ThinkTwice devotes every second update to refinement, it covers half as many distinct training problems as the baselines within the same budget, with each sampled problem receiving both a solving and a refinement pass. Our comparisons therefore measure the value of spending half of the update budget on refinement rather than on additional problems.

Our evaluations are two-fold. In Section 4.2, we test the reasoning capability of ThinkTwice and baselines with direct prompting as a single-turn generation. In Section 4.3, we evaluate the refinement capability using the multi-turn chat format, testing models on refining both their own generations and also those produced by others.

4.2 Reasoning

For each benchmark problem, we sample n=32 independent solutions for Qwen3-4B and OLMo3-7B. Following Chen et al. (2021), we report pass@4, the estimated probability that at least one of 4 sampled solutions is correct. Figure 6 (Appendix C) reports pass@k for k from 1 to 32: at k = 1, ThinkTwice is roughly on par with DAPO, and its advantage grows with k, most clearly on AIME with Qwen3-4B.

Table 1 reports the reasoning performance on both models. For Qwen3-4B, ThinkTwice achieves the highest average score of 65.57%, outperforming all competitive baselines. The gains are most pronounced on the most challenging AIME benchmark, where ThinkTwice reaches 44.11% compared to 39.06% for GRPO. Across the remaining benchmarks, ThinkTwice is consistently the best or second-best method. For OLMo3-7B, the overall trend mirrors that of Qwen3-4B, with ThinkTwice achieving the highest average score. Notably, these improvements are obtained on direct prompting alone, before any self-refinement step is applied, which indicates that the refinement training phase in ThinkTwice itself strengthens the model's reasoning capability.

[3] https://github.com/verl-project/verl

Table 2: Self-refinement performance (pass@4, ↑) across five mathematical reasoning benchmarks. **Bold** and underline denote the best and second-best results per dataset within each model. † denotes training-free methods.

Method	AIME	AMC	MATH500	Minerva	OlympiadBench	Average
Qwen3-4B						
Base Model	45.25	78.10	92.82	40.81	63.52	64.10
Reflexion†	38.47	73.17	91.48	40.87	60.89	60.98
Self-Refine†	50.37	82.40	93.86	41.19	66.33	66.83
GRPO	48.91	81.86	93.78	42.90	69.67	67.42
DrGRPO	46.98	82.66	94.46	**44.84**	71.75	68.14
DAPO	52.88	84.77	95.33	43.67	72.69	69.87
ThinkTwice	**60.43**	**85.54**	**95.70**	43.93	**73.78**	**71.88**
OLMo3-7B						
Base Model	39.71	78.18	91.75	41.58	66.14	63.47
Reflexion†	37.38	72.18	91.29	41.48	62.36	60.94
Self-Refine†	47.34	83.81	93.24	42.35	68.30	67.01
GRPO	46.04	84.48	93.76	41.08	66.54	66.38
DrGRPO	45.24	82.32	93.54	42.75	69.81	66.73
DAPO	44.26	84.55	93.33	42.81	70.57	67.10
ThinkTwice	**49.33**	**87.06**	**94.66**	**44.33**	**71.38**	**69.35**

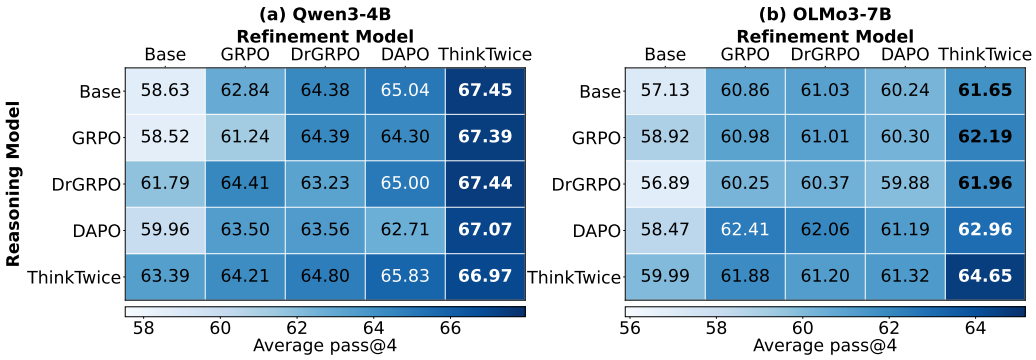

Figure 3: Cross-model refinement evaluation (average pass@4, ↑). Rows denote the backbone reasoning model; columns denote the refinement model.

4.3 Refinement

Self-Refinement. We first evaluate the end-to-end reason-then-refine pipeline: for each problem, we sample n=32 base solutions, refine each with greedy decoding (thus yielding n=32 refined solutions), and report pass@4 over the refined solutions. Table 2 reports results on Qwen3-4B and Olmo3-7B. For Qwen3-4B, ThinkTwice achieves the highest average of 71.88%, outperforming DAPO by 2.0 percentage points and GRPO by 4.5 points. On AIME, ThinkTwice reaches 60.43%, a gain of over 11 percentage points above GRPO. For OLMo3-7B, the pattern is consistent: ThinkTwice again achieves the highest average self-refinement score, outperforming all baselines on every benchmark. Meanwhile, ThinkTwice also significantly outperforms training-free baselines[4]. The full self-refinement pass@k curves are provided in Figure 7 at Appendix C.

Cross-Model Refinement. To isolate refinement capability from base solution quality, we evaluate each model as a refinement model applied to every other model's base solutions. For each problem, we generate a single base solution with greedy decoding from the reasoning model, then sample 4

[4]Even under our favorable implementation (details in Appendix B.2), Reflexion appears less suited to offline math refinement, because retrying from scratch after reflection can discard useful local corrections.

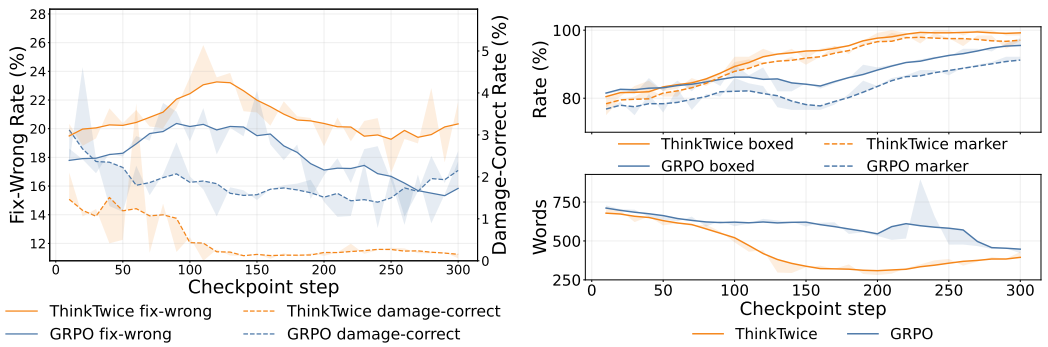

Figure 4: Training dynamics of refinement across checkpoints. Left (a): transition metrics on the training set across checkpoints. Right (b): formatting and length metrics across checkpoints during training. Top: boxed-answer and final-answer marker rates; bottom: average response length for self-refinement on correct-only base solutions.

refinements from the refinement model and report the accuracy after refinement. This protocol fixes the input and attributes all variance to the model's refinement capabilities. Figure 3 reports the results averaged across all five benchmarks. ThinkTwice as the refinement model (the rightmost column) achieves the highest score regardless of which model produced the base solution, demonstrating that its refinement capability generalizes beyond its own outputs.

Taken together, these results show that ThinkTwice improves both reasoning and self-refinement within a single training framework, and that these gains persist across two different types and sizes of LLMs.

5 Discussion

5.1 Training Dynamics of ThinkTwice

We provide quantitative and qualitative analyses for understanding the learning dynamics of ThinkTwice. Our central observation is that ThinkTwice gives rise to an implicit *rectify-then-fortify* dynamic: refinement predominantly corrects errors early in training, and gradually shifts toward preserving and polishing already-correct solutions as the base policy improves.

Transition Metrics. To characterize this dynamic, we first define two conditional transition rates measured on the training set throughout the course of training. Let *fix-wrong* (%) denote the fraction of incorrect base solutions that become correct after refinement, and *damage-correct* (%) denote the fraction of correct base solutions that become incorrect after refinement. Figure 4a plots both metrics across training checkpoints for ThinkTwice and the GRPO baseline.

Early Training: Rectification via Refinement. Early in training, incomplete derivations or unproductive solution paths can cause base attempts to fail. Refinement gives the model a second attempt on the same problem. Figure 4a shows that ThinkTwice maintains a higher *fix-wrong* rate than GRPO throughout training, peaking around the midpoint. Recovering correct solutions on problems near the model's capability boundary yields a more *rectified* reward signal than base reasoning alone.

Late Training: Fortification via Refinement. As base accuracy improves, refinement shifts from error correction to solution preservation. Figure 4a shows that ThinkTwice's *damage-correct* rate drops near zero in the second half of training, while GRPO's rate remains more than 5 times higher than ThinkTwice's best checkpoint. Correct refined solutions also become shorter over training (Figure 4b, Bottom), consistent with removing exploratory clutter from already-correct solutions. Refined outputs have higher rates of boxed answers and *Final Answer* markers than GRPO despite receiving no format reward (Figure 4b, Top).

Qualitative Evidence. Appendix D illustrates *route switching*, where refinement replaces an unsuccessful solution path; *solution completion*, where it finishes a promising but incomplete trace; and

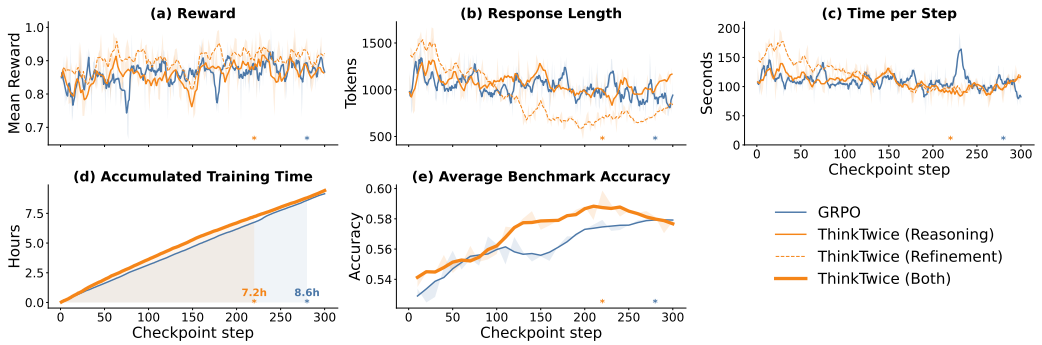

Figure 5: Training-time cost and dynamics of ThinkTwice compared with GRPO. * marks best checkpoint step of models. (a) Mean reward. (b) Response length. (c) Wall-clock time per update. (d) Accumulated training time. (e) Within-checkpoint macro average benchmark accuracy. Solid orange denotes ThinkTwice base updates, solid blue denotes GRPO, and dashed orange denotes ThinkTwice refinement updates when applicable.

late-stage fortification, where it shortens an already-correct solution without changing the answer. These cases support the observed transition from correction to preservation during training.

5.2 Training Cost of ThinkTwice

Figure 5 compares the training stability and cost of ThinkTwice and GRPO on Qwen3-4B.

Panel (a) shows that mean reward remains comparably stable to GRPO. In Panel (b), base response lengths stay in the same range as GRPO, while refinement outputs shorten as training proceeds, consistent with the dynamics in Section 5.1.

Panel (c) reports wall-clock time per update.[5] Refinement updates initially cost more because their prompts include the earlier solution. As refined responses shorten, the cost gap narrows. By step 300, ThinkTwice uses only 3% more total wall-clock time than GRPO (9.42 hours vs 9.15 hours; Panel (d)). It reaches its best checkpoint in 16% less time (7.2h vs 8.6h), requiring fewer total steps (220 vs 280). Panel (e) shows faster improvement in early and middle training and higher benchmark accuracy for most of the trajectory.

6 Conclusion

We introduced ThinkTwice, a two-phase RLVR framework that jointly optimizes reasoning and self-refinement using the same binary correctness reward in both phases, without correctness signals, critique annotations, or external verifiers. Across five mathematical reasoning benchmarks and two model families, ThinkTwice consistently outperforms competitive online policy optimization baselines on both direct reasoning and self-refinement. Our quantitative and qualitative analyses of the training dynamics of ThinkTwice reveal an implicit rectify-then-fortify curriculum that yields a more rectified reward signal, while adding only 3% to total wall-clock time at matched update counts over the GRPO baseline we build upon.

Table 3: Compute to reach the best checkpoint.

	GRPO	ThinkTwice
Checkpoint step	280	220
Updates (solve/refine)	280/0	110/110
Rollouts	71.7K	56.3K
Unique problems	7.5K	3.5K
GPU-hours	17.2	14.4

More broadly, these results show that self-refinement, a basic building block of self-improving systems, can be learned from the same sparse outcome reward that RLVR already assumes and then run at inference without any external signal. Our training uses a single refinement turn; whether the gains compound over repeated turns, and whether the approach carries over to other verifiable domains such as code generation, are natural directions for future work.

[5] All timing results are measured on 2×H100 80GB GPUs. Hyperparameter details are in Appendix B.1.

ETHICS STATEMENT

This work focuses on improving mathematical reasoning capabilities of large language models through reinforcement learning with verifiable rewards. Our method uses publicly available datasets and models, and does not involve human subjects, private data, or dual-use concerns beyond those inherent to general-purpose language model research.

REPRODUCIBILITY STATEMENT

We provide comprehensive implementation details to facilitate reproduction of our results. Appendix B.1 includes the full training workflow with code-level illustrations, the exact refinement instruction, hyperparameter configurations (Table 5), and dataset sources with links. Appendix B.2 details the implementation of all training-free baselines. All experiments use publicly available base models and datasets.

AI USE STATEMENT

Generative AI assisted with writing, reference checks, figures, and formatting. The authors are responsible for the final content.

REFERENCES

Collin Burns, Pavel Izmailov, Jan Hendrik Kirchner, Bowen Baker, Leo Gao, Leopold Aschenbrenner, Yining Chen, Adrien Ecoffet, Manas Joglekar, Jan Leike, et al. Weak-to-strong generalization: Eliciting strong capabilities with weak supervision. *arXiv preprint arXiv:2312.09390*, 2023.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Xiangxiang Chu, Hailang Huang, Xiao Zhang, Fei Wei, and Yong Wang. Gpg: A simple and strong reinforcement learning baseline for model reasoning. *arXiv preprint arXiv:2504.02546*, 2025.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

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

Wei Fu, Jiaxuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, et al. Areal: A large-scale asynchronous reinforcement learning system for language reasoning. *arXiv preprint arXiv:2505.24298*, 2025.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. Critic: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*, 2023.

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

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3828–3850, 2024.

Jujie He, Jiacai Liu, Chris Yuhao Liu, Rui Yan, Chaojie Wang, Peng Cheng, Xiaoyu Zhang, Fuxiang Zhang, Jiacheng Xu, Wei Shen, et al. Skywork open reasoner 1 technical report. *arXiv preprint arXiv:2505.22312*, 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.

Jingcheng Hu, Yinmin Zhang, Qi Han, Daxin Jiang, Xiangyu Zhang, and Heung-Yeung Shum. Open-reasoner-zero: An open source approach to scaling up reinforcement learning on the base model. *arXiv preprint arXiv:2503.24290*, 2025.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*, 2023.

Yuhua Jiang, Yuwen Xiong, Yufeng Yuan, Chao Xin, Wenyuan Xu, Yu Yue, Qianchuan Zhao, and Lin Yan. Pag: Multi-turn reinforced llm self-correction with policy as generative verifier. *arXiv preprint arXiv:2506.10406*, 2025.

Difan Jiao, Raghav Singhal, Robert West, and Ashton Anderson. Tandem reinforcement learning with verifiable rewards. *arXiv preprint arXiv:2606.28166*, 2026. URL https://arxiv.org/abs/2606.28166.

Yizhu Jiao, Jiaqi Zeng, Julien Veron Vialard, Oleksii Kuchaiev, Jiawei Han, and Olivier Delalleau. Think twice: Branch-and-rethink reasoning reward model. *arXiv preprint arXiv:2510.23596*, 2025.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, et al. Training language models to self-correct via reinforcement learning. *arXiv preprint arXiv:2409.12917*, 2024.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in neural information processing systems*, 35:3843–3857, 2022.

Moxin Li, Wenjie Wang, Fuli Feng, Fengbin Zhu, Qifan Wang, and Tat-Seng Chua. Think twice before trusting: Self-detection for large language models through comprehensive answer reflection. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 11858–11875, 2024.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The twelfth international conference on learning representations*, 2023.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025.

Ruotian Ma, Peisong Wang, Cheng Liu, Xingyan Liu, Jiaqi Chen, Bang Zhang, Xin Zhou, Nan Du, and Jia Li. S2r: Teaching llms to self-verify and self-correct via reinforcement learning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 22632–22654, 2025.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Lukáš Mikula, Michal Štefánik, Marek Petrovič, and Petr Sojka. Think twice: Measuring the efficiency of eliminating prediction shortcuts of question answering models. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2179–2193, 2024.

Team Olmo, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, et al. Olmo 3. *arXiv preprint arXiv:2512.13961*, 2025.

Hoang Phan, Victor Li, and Qi Lei. Think twice, generate once: Safeguarding by progressive self-reflection. *arXiv preprint arXiv:2510.01270*, 2025.

George Polya. *How to solve it: A new aspect of mathematical method*. Princeton university press, 1945.

Yushan Qian, Bo Wang, Shangzhao Ma, Wu Bin, Shuo Zhang, Dongming Zhao, Kun Huang, and Yuexian Hou. Think twice: A human-like two-stage conversational agent for emotional response generation. *arXiv preprint arXiv:2301.04907*, 2023.

Yuxiao Qu, Tianjun Zhang, Naman Garg, and Aviral Kumar. Recursive introspection: Teaching language model agents how to self-improve. *Advances in Neural Information Processing Systems*, 37:55249–55285, 2024.

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

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, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Rohan Surana, Gagan Mundada, Xunyi Jiang, Chuhan Wang, Zhenwei Tang, Difan Jiao, Zihan Huang, Yuxin Xiong, Junda Wu, Sheldon Yu, et al. Generate, filter, control, replay: A comprehensive survey of rollout strategies for LLM reinforcement learning. *arXiv preprint arXiv:2605.02913*, 2026. URL https://arxiv.org/abs/2605.02913.

Xiaoyu Tian, Sitong Zhao, Haotian Wang, Shuaiting Chen, Yunjie Ji, Yiping Peng, Han Zhao, and Xiangang Li. Think twice: Enhancing llm reasoning by scaling multi-round test-time thinking. *arXiv preprint arXiv:2503.19855*, 2025.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439, 2024.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2022. URL https://arxiv.org/abs/2203.11171. ICLR 2023.

Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin Choi. Generating sequences by learning to self-correct. *arXiv preprint arXiv:2211.00053*, 2022.

Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Shengping Liu, Bin Sun, Kang Liu, and Jun Zhao. Large language models are better reasoners with self-verification. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 2550–2575, 2023.

Alex Wilf, Sihyun Lee, Paul Pu Liang, and Louis-Philippe Morency. Think twice: Perspective-taking improves large language models' theory-of-mind capabilities. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8292–8308, 2024.

Zhiheng Xi, Dingwen Yang, Jixuan Huang, Jiafu Tang, Guanyu Li, Yiwen Ding, Wei He, Boyang Hong, Shihan Do, Wenyu Zhan, et al. Enhancing llm reasoning via critique models with test-time and training-time supervision. *arXiv preprint arXiv:2411.16579*, 2024.

Wei Xiong, Hanning Zhang, Chenlu Ye, Lichang Chen, Nan Jiang, and Tong Zhang. Self-rewarding correction for mathematical reasoning. *arXiv preprint arXiv:2502.19613*, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, et al. DAPO: An open-source LLM reinforcement learning system at scale, 2025a. URL https://arxiv.org/abs/2503.14476.

Tianshu Yu, Chao Xiang, Mingchuan Yang, Pei Ke, Bosi Wen, Cunxiang Wang, Jiale Cheng, Li Zhang, Xinyu Mu, Chuxiong Sun, et al. Training language model to critique for better refinement. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 26760–26804, 2025b.

Lifan Yuan, Wendi Li, Huayu Chen, Ganqu Cui, Ning Ding, Kaiyan Zhang, Bowen Zhou, Zhiyuan Liu, and Hao Peng. Free process rewards without process labels. *arXiv preprint arXiv:2412.01981*, 2024.

Guanning Zeng, Zhaoyi Zhou, Daman Arora, and Andrea Zanette. Shrinking the variance: Shrinkage baselines for reinforcement learning with verifiable rewards. *arXiv preprint arXiv:2511.03710*, 2025a.

Weihao Zeng, Yuzhen Huang, Qian Liu, Wei Liu, Keqing He, Zejun Ma, and Junxian He. Simplerl-zoo: Investigating and taming zero reinforcement learning for open base models in the wild. *arXiv preprint arXiv:2503.18892*, 2025b.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. ReST-MCTS*: LLM self-training via process reward guided tree search, 2024a. URL https://arxiv.org/abs/2406.03816. NeurIPS 2024.

Fuxiang Zhang, Jiacheng Xu, Chaojie Wang, Ce Cui, Yang Liu, and Bo An. Incentivizing llms to self-verify their answers. *arXiv preprint arXiv:2506.01369*, 2025a.

Xiaoying Zhang, Yipeng Zhang, Hao Sun, Kaituo Feng, Chaochao Lu, Chao Yang, and Helen Meng. Critique-grpo: Advancing llm reasoning with natural language and numerical feedback. *arXiv preprint arXiv:2506.03106*, 2025b.

Yunxiang Zhang, Muhammad Khalifa, Lajanugen Logeswaran, Jaekyeom Kim, Moontae Lee, Honglak Lee, and Lu Wang. Small language models need strong verifiers to self-correct reasoning. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 15637–15653, 2024b.

Xutong Zhao, Tengyu Xu, Xuewei Wang, Zhengxing Chen, Di Jin, Liang Tan, Zishun Yu, Zhuokai Zhao, Yun He, Sinong Wang, et al. Boosting llm reasoning via spontaneous self-correction. *arXiv preprint arXiv:2506.06923*, 2025a.

Yuzhong Zhao, Yue Liu, Junpeng Liu, Jingye Chen, Xun Wu, Yaru Hao, Tengchao Lv, Shaohan Huang, Lei Cui, Qixiang Ye, et al. Geometric-mean policy optimization. *arXiv preprint arXiv:2507.20673*, 2025b.

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

A Additional Related Work Comparison

Table 4: A compact comparison of ThinkTwice with related work. We compare whether methods use reinforcement learning, require supervision beyond final-answer correctness, and rely on an additional verifier, teacher, or explicit critique channel. Entries summarize the dominant tendency of each family.

Family	Representative methods	RL?	Extra supervision?	Extra model / critique	Core characteristic
One-pass RLVR	GRPO, DeepSeek-R1, DAPO, Dr. GRPO	Yes	No	No	Trains first-pass reasoning only; no learned revision policy.
Collaborative RLVR	Tandem RL (TRL) (Jiao et al., 2026)	Yes	No	Frozen junior partner	Trains a senior on jointly generated reasoning trajectories to improve partner compatibility.
Training-free refinement	Self-Refine, Reflexion, self-verification prompting, CRITIC, self-consistency	No	No during training	No, or optional tools	Uses extra test-time feedback or sampling, but refinement is not learned.
Training-based refinement	Cobbe verifier, Let's Verify, Math-Shepherd, Free Process Rewards, PRIME, AutoMathCritique, CFT, Recursive Introspection, Self-rewarding Correction, SPOC	Mixed	Often yes	Often yes or partial	Usually relies on process labels, critique data, synthetic correction traces, or stronger verifiers.
RL-based refinement	SCoRe, S^2R, PAG, Self-Verify, Critique-GRPO	Yes	Usually yes	Sometimes	Closest prior line, but typically adds an explicit verification or critique objective, or a reward that scores the first attempt.
ThinkTwice	Two-phase RLVR with a shared policy	**Yes**	**No**	**No**	**Same correctness reward in solve and revise phases; revision conditions only on the model's prior answer plus a generic review instruction.**

Position of ThinkTwice in the literature. Table 4 extends the overview in Section 2 with a broader comparison between ThinkTwice and prior work. We organize the literature into five families: one-pass RLVR methods, collaborative RLVR methods, training-free self-refinement methods, training-based self-refinement methods, and RL-based self-refinement methods. The comparison highlights that ThinkTwice is closest to the RL-based self-refinement line, but differs in using a shared policy and the same sparse final-answer correctness reward in both the solve and revise phases, without process labels, critique annotations, correctness hints, or a reward that depends on whether the initial answer is correct.

Rollout construction. Rollout design determines the trajectories from which an RL system learns (Surana et al., 2026). Tandem Reinforcement Learning (TRL) (Jiao et al., 2026) trains a senior model on reasoning trajectories jointly generated with a frozen junior, using standard GRPO and final-answer rewards to improve partner compatibility. ThinkTwice uses successive solve-and-refine rollouts from a shared policy, making revision of a complete prior answer the second training task.

Other second-pass methods. Prior work has explored second-pass reasoning, reflection, and related reconsideration strategies in a range of settings, including emotional dialogue generation (Qian et al., 2023), Theory-of-Mind prompting via *SimToM* (Wilf et al., 2024), shortcut analysis in question answering (Mikula et al., 2024), LLM self-detection via *T3* (Li et al., 2024), *Multi-round Thinking* for inference-time scaling (Tian et al., 2025), *Progressive Self-Reflection* for safety (Phan et al., 2025), and *branch-and-rethink* for reward modeling (Jiao et al., 2025). Although these works share the high-level intuition that revisiting an initial response can help, they differ substantially in task, objective, and supervision. In contrast, our ThinkTwice jointly trains a single reasoning policy with RL to perform both first-pass reasoning and second-pass refinement using the same sparse final-answer correctness reward in both phases, without critique annotations, external verifiers, or a separate reward-model objective.

B REPRODUCIBILITY

B.1 THINKTWICE IMPLEMENTATION DETAILS

B.1.1 DATASETS

We use the MATH dataset [6] for training, which consists of 7,500 problems spanning difficulty levels 1–5 across seven subjects. For evaluation, we use a test set of 1,526 problems from five widely used mathematical reasoning benchmarks:

- **MATH500** [7]: 500 problems from the MATH test set.
- **OlympiadBench** [8]: 581 problems from olympiad-level competitions.
- **Minerva Math** [9]: 272 problems from high school math competitions.
- **AIME**: 90 problems from AIME 2022–2024 [10].
- **AMC**: 83 problems from the AMC 10/12 competitions [11].

Each problem is formatted with a task instruction requiring the model to output its final answer in boxed format (`\boxed{}`). We check the final answer against the ground truth with a boxed answer grader and Huggingface Math-Verify.

B.1.2 A DETAILED ILLUSTRATION OF THINKTWICE TRAINING WORKFLOW

We provide a concrete implementation-style walkthrough of ThinkTwice training, bridging the Algorithm 1 pseudocode with the actual code execution flow. The following uses Python-like notation containing the core training logic from `ray_trainer.py`.

The two-phase training. The implementation follows the two-phase structure in Algorithm 1. When we activate ThinkTwice training, the main training loop alternates between reasoning and refinement steps within each iteration:

```
for epoch in range(total_epochs):
    for batch in dataloader:
        # Phase 1: Reasoning (with refinement candidate preparation)
        pending_refinement = run_step(batch, allow_refinement=True)

        # Phase 2: Refinement (if candidates selected)
        if pending_refinement:
            run_step(pending_refinement, allow_refinement=False)
```

When `allow_refinement=True`, `run_step` internally performs candidate selection after reward computation and policy update:

```
num_select = len(batch) // repeat_times  # One per prompt
refinement_indices = select_refinement_indices(
    uids, responses, is_correct, num_select,
    selection_mode='random'  # for random selection from base solutions
)
selected_base = base_batch.select_idxs(refinement_indices)
selected_gen = gen_batch_output.select_idxs(refinement_indices)
# yielding (selected_base, selected_gen) for refinement prompt
```

The refinement prompts are then constructed from `selected_gen` (containing the generated responses) and `selected_base` (containing the original prompts):

[6] https://huggingface.co/datasets/hendrycks/math
[7] https://huggingface.co/datasets/HuggingFaceH4/MATH-500
[8] https://huggingface.co/datasets/afraamn/olympiadbench_math_textonly
[9] https://huggingface.co/datasets/math-ai/minervamath
[10] https://huggingface.co/datasets/AI-MO/aimo-validation-aime
[11] https://huggingface.co/datasets/AI-MO/aimo-validation-amc

Category	Hyperparameter
GRPO Training	
Learning rate	1×10^{-6}
PPO clip ratio	0.2
Max response length	3000
Train batch size	32
PPO mini batch size	8
Group size (G)	8
Entropy coefficient	0.0
KL penalty in reward	Disabled
KL coefficient (β)	0.0
Refinement Training	
Refinement steps	2 (one reasoning + one refinement)
Refinement selection mode	random
Generation	
Temperature	1.0 (train), 0.0 (val)
Top-p	1.0
Top-k	-1
Max model length	8192

Table 5: Key hyperparameters for ThinkTwice training.

```
messages = [
    {'role': 'user', 'content': problem}, # raw prompt
    {'role': 'assistant', 'content': base_solution}, # from Phase 1
    {'role': 'user', 'content': refinement_INSTRUCTION}
]
# messages is then packaged into pending_refinement, then passed to
    run_step() for Phase 2
```

with the refinement instruction as:

> Follow this instruction, carefully review your previous solution:
> 1. Go through each calculation step-by-step. Check if there are any errors in calculations, logic, or problem understanding.
> 2. If you find any mistakes, explicitly point out what was wrong and explain the correct approach.
> 3. If the solution is already correct, verify each step and explain it more clearly.
> 4. Finally, after finishing the review, provide your refined solution and answer.

The refinement instruction is task-agnostic and contains no correctness signals, ensuring the model learns self-refinement without external supervision.

B.1.3 Hyperparameter Configuration

Table 5 summarizes the key hyperparameters for ThinkTwice training. For every method, we keep the checkpoint with the highest average accuracy on a fixed random 20% subset of each benchmark and stop training after 100 steps without improvement.

B.2 Implementation of Training-free Refinement Baselines

We compare our trained models against four non-training test-time baselines: *base reasoning*, *one-step refinement*, and two iterative methods inspired by Self-Refine (Madaan et al., 2023) and Reflexion (Shinn et al., 2023). All baselines are evaluated in *non-thinking* mode. To isolate the effect of the inference procedure from prompt engineering, we keep the refinement instruction fixed across all refinement-based baselines.

Prompt formatting. Each evaluation example is rendered with the model's chat template and an added generation prompt. For OLMo-3-7B-Instruct, the tokenizer template automatically

prepends a default system message. In practice, we remove only this automatically injected prefix before decoding, so that the model is evaluated without the template-provided system instruction. For one-step refinement, SELF-REFINE, and the memory-generation stage of REFLEXION, we use the same refinement prompt as shown in Appendix B.1.

base reasoning. For each problem x_i, we sample $n = 32$ candidate solutions

$$y_{i,1}^{(0)}, y_{i,2}^{(0)}, \ldots, y_{i,n}^{(0)}$$

from the model using temperature 0.7, top-$p = 0.8$, top-$k = 20$, and a maximum generation length of 3000 tokens. All second-stage generations (refinement, refinement, and retry) are decoded greedily. Each initial sample defines one *branch*. Iterative updates occur only *within* a branch, and only the final answer of that branch is scored. Consequently, iterative baselines do not receive extra $\mathrm{pass@}k$ opportunities beyond the original n branches.

One-step refinement. Given a sampled solution $y_{i,j}^{(0)}$, we construct a second-round prompt containing the original problem, the sampled answer as an assistant turn, and the shared refinement instruction. We then decode a single refined solution $\hat{y}_{i,j}$ greedily. The set

$$\{\hat{y}_{i,j}\}_{j=1}^{n}$$

constitutes the refinement baseline for problem i.

SELF-REFINE. We implement SELF-REFINE as repeated refinement-and-refinement within the same branch, while keeping the refinement instruction identical to the one-step refinement baseline. Starting from $y_{i,j}^{(0)}$, we iteratively apply

$$y_{i,j}^{(t+1)} = \mathcal{M}\Big(x_i, y_{i,j}^{(t)}, r\Big), \qquad t = 0, \ldots, T-1,$$

where $\mathcal{M}$ denotes the chat model and r denotes the fixed refinement instruction. We score only the final output $y_{i,j}^{(T)}$. Under this controlled prompt design, the case $T = 1$ is identical to one-step refinement; therefore, when reporting SELF-REFINE as a distinct baseline, we use $T \geq 2$.

REFLEXION. For REFLEXION, we again begin from a base sample $y_{i,j}^{(0)}$, but use the refinement stage to produce a short verbal memory rather than a final revised answer. Let $m_{i,j}^{(1)}$ denote the first refinement note. The model then retries the original problem from scratch while conditioning on the accumulated memory:

$$\tilde{y}_{i,j}^{(t+1)} = \mathcal{M}\Big(x_i, m_{i,j}^{(\leq t)}\Big).$$

We report the final retry output after the prescribed number of retries. In our default setting, we use one retry so that the test-time cost remains comparable to one-step refinement and two-step SELF-REFINE. Ground-truth answers are never revealed during inference; they are used only for offline evaluation.

Evaluation Protocol. A branch is marked correct if its final answer matches the ground-truth solution under the same boxed-answer grader used throughout evaluation. Let c_i be the number of correct branch finals among the $n = 32$ branches for problem i. Similarly, we estimate $\mathrm{pass@}k$ following Chen et al. (2021). This estimator allows all reported values of k to be obtained from a single set of sampled generations, without rerunning inference separately for each k.

Reproducibility note. All baselines use the same random seed and the same underlying evaluation set. The only difference between methods is how each branch is post-processed at test time: no revision for base reasoning, one revision for refinement, repeated revision for SELF-REFINE, and retry conditioned on verbal memory for REFLEXION.

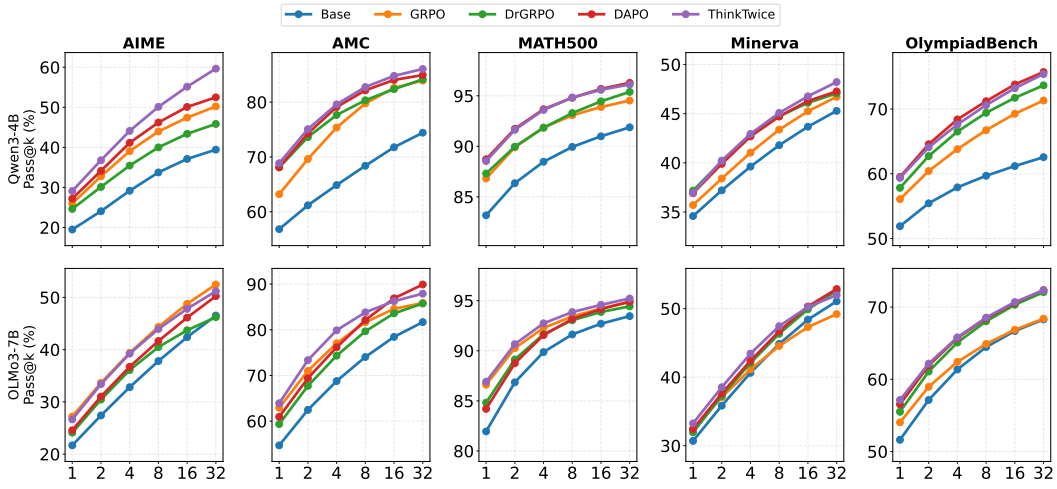

Figure 6: Reasoning pass@k curves across five mathematical reasoning benchmarks for Qwen3-4B (top) and OLMo3-7B (bottom).

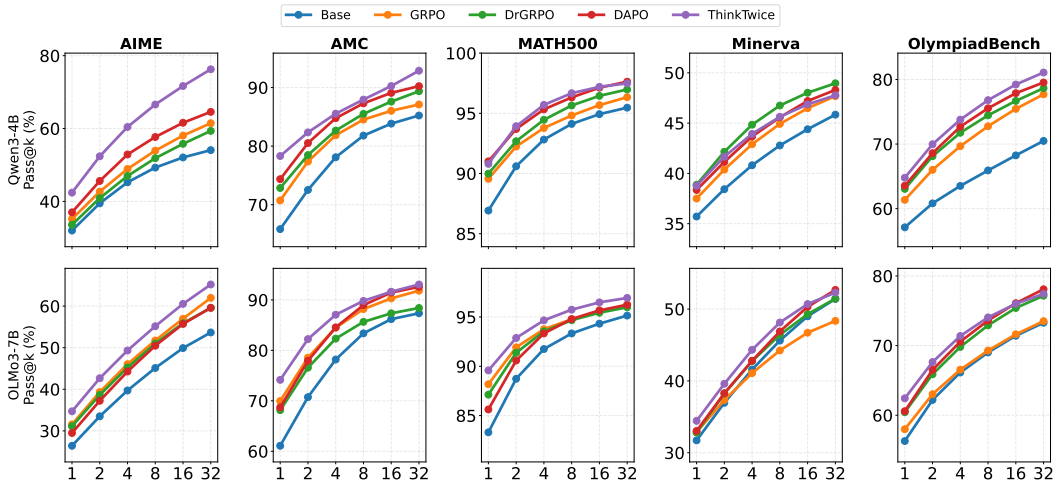

Figure 7: Self-refinement pass@k curves across five mathematical reasoning benchmarks for Qwen3-4B (top) and OLMo3-7B (bottom).

C Additional Results

C.1 Full Pass@k Curves

We provide the full pass@k curves for both reasoning and self-refinement evaluation, complementing the pass@4 results reported in Tables 1 and 2. Figure 6 shows reasoning performance and Figure 7 shows self-refinement performance, both across $k \in \{1, 2, 4, 8, 16, 32\}$ for all five benchmarks and both model families.

For reasoning (Figure 6), ThinkTwice is the best or second-best method at nearly every k on both Qwen3-4B and OLMo3-7B, and its advantage is most pronounced on AIME for Qwen3-4B, where the gap widens steadily as k increases. For self-refinement (Figure 7), the separation is even more striking: ThinkTwice maintains a clear lead on AIME across the entire k range, reaching over 76% pass@32 on Qwen3-4B compared to roughly 65% for the next-best baseline. Across both models, the curves confirm that ThinkTwice's improvements are not artifacts of a particular k value but hold consistently across the pass@k spectrum.

Table 6: Reasoning (pass@8, ↑) and self-refinement (pass@4, ↑) performance of ThinkTwice across five mathematical reasoning benchmarks.

Evaluation	AIME	AMC	MATH500	Minerva	Olympiad	Average
Qwen3-4B						
Reasoning	50.11	82.75	94.82	45.07	70.59	68.67
Refinement	60.43	85.54	95.70	43.93	73.78	71.88
OLMo3-7B						
Reasoning	43.96	83.82	93.86	47.44	68.55	67.53
Refinement	49.33	87.06	94.66	44.33	71.38	69.35

C.2 Self-Refinement versus Additional Sampling

We test whether a generation spent on refinement raises pass@k more than a generation spent on a new independent sample. The self-refinement evaluation in Table 2 uses two generations for each sampled solution, one to reason and one to refine, so its gain over reasoning could come from the extra generation alone. Self-refinement pass@4 and reasoning pass@8 both use eight generations per problem, and Table 6 compares them for ThinkTwice with values read from Figures 6 and 7. Self-refinement scores higher on four of the five benchmarks for both models, and its average exceeds reasoning pass@8 by 3.2 points on Qwen3-4B and 1.8 points on OLMo3-7B. The gain is largest on AIME, the most challenging benchmark, and Minerva is the exception. At an equal number of generations, refinement therefore raises the pass@k of ThinkTwice more than additional sampling does.

D Case Studies

The three recurring mechanisms we wanted to document are: (i) **route switching**, where refinement abandons a bad search path and finds a better method; (ii) **solution completion**, where the base trace is promising but unfinished and refinement closes the argument; and (iii) **late-stage exploitation**, where the base trace is already correct but refinement removes exploratory clutter, normalizes the final answer, and presents a shorter proof. Table 7 summarizes the selected cases.

Case	Sample	Checkpoints	Why it matters
A	481	100, 240	Same problem shows both phases of the story: early refinement discovers the missing telescoping structure; later refinement gives a much shorter version of the now-correct base proof.
B	497	110, 250	refinement completes a stalled geometric derivation early, then later preserves the method while converting a long decimal-heavy trace into a shorter exact solution.
C	194	30, 220	Early refinement repairs a concrete derivative mistake; later refinement turns a wandering but correct proof into a short direct argument.
D	425	100, 220	Geometry case: early refinement recomputes faulty vector algebra; later refinement keeps the invariant but removes most of the symbolic overhead.

Table 7: Case-study map for the detailed qualitative appendix.

D.1 CASE A: SAMPLE 481 — EARLY STRUCTURAL DISCOVERY, LATE EXPLOITATION ON THE *same* PROBLEM

Problem and role in the appendix

Problem (MATH500, level 5, sample 481). Let n be a positive integer. Simplify

$$\frac{(2^4+\frac{1}{4})(4^4+\frac{1}{4})\cdots[(2n)^4+\frac{1}{4}]}{(1^4+\frac{1}{4})(3^4+\frac{1}{4})\cdots[(2n-1)^4+\frac{1}{4}]}$$

Ground truth: $\boxed{8n^2+4n+1}$.

Why this case is useful. This is the clearest single-problem narrative we found. At step 100 the ThinkTwice reasoning trace is wrong; the refinement keeps the same factorization but discovers the missing telescoping relation and solves the problem. By step 240 the *same* problem has become a late-stage exploitation example: the ThinkTwice reasoning is now correct but still very long, while the refinement keeps the proof and compresses it aggressively. The pure-GRPO baseline is informative at both checkpoints: it reaches the right refinement answer at step 100 only after a longer pattern-checking route, and by step 240 its base trace is still wrong while its refinement remains much longer than ThinkTwice's.

Checkpoint	Run	Base output	refinement output
Step 100	ThinkTwice	WRONG; 1222 words; last boxed value $\boxed{85}$	CORRECT; 686 words; $\boxed{8n^2+4n+1}$
	Pure GRPO	WRONG; 1308 words; no final boxed general form	CORRECT; 892 words; $\boxed{8n^2+4n+1}$
Step 240	ThinkTwice	CORRECT; 1222 words; $\boxed{8n^2+4n+1}$	CORRECT; 358 words; $\boxed{8n^2+4n+1}$
	Pure GRPO	WRONG; 1243 words; no final boxed general form	CORRECT; 1067 words; $\boxed{8n^2+4n+1}$

Table 8: Case A summary. The same problem exhibits both early correction and late-stage compression.

EARLY CHECKPOINT (STEP 100): REFINEMENT FINDS THE MISSING TELESCOPING ARGUMENT

The ThinkTwice reasoning answer already has the right local algebraic move,

$$x^4+\tfrac{1}{4}=\left(x^2-x+\tfrac{1}{2}\right)\left(x^2+x+\tfrac{1}{2}\right),$$

but it never turns that factorization into a general product formula. Instead, the trace drifts into pattern-matching on small values of n:

ThinkTwice reasoning, step 100 (wrong; 1222 words)

"But maybe we can look for a telescoping pattern? Let's compute the expression for small values of n and see if we can spot a pattern." It checks $n=1,2,3$ and ends with the last computed value $\boxed{85}$, which is the $n=3$ instance rather than a general closed form.

The refinement does *not* restart from scratch. Instead, it keeps the factorization and adds the structural observation that the base trace missed. After simplifying a single term to

$$\frac{8k^2+4k+1}{8k^2-12k+5},$$

it makes the crucial recurrence explicit:

ThinkTwice refinement, step 100 (correct; 686 words)

"So the denominator of term k is equal to the numerator of term $k-1$. Therefore, the entire product telescopes." It then rewrites the product as

$$\prod_{k=1}^{n} \frac{b_{k+1}}{b_k} = \frac{b_{n+1}}{b_1}, \qquad b_k = 8k^2 - 12k + 5,$$

and concludes with $\boxed{8n^2 + 4n + 1}$.

The same-step pure-GRPO baseline refinement also reaches the right answer, but by a looser and longer route. It explicitly computes several small cases before inferring the formula:

Pure-GRPO refinement, step 100 (correct; 892 words)

"Step 4: Look for telescoping pattern. We now compute this product for small n to detect a pattern." Later it writes "Perfect! So the entire product simplifies to $8n^2 + 4n + 1$."

The important qualitative difference here is not that ThinkTwice is the only system that can solve the problem at this checkpoint; rather, ThinkTwice's refinement is already more *structural*. The base gets trapped in empirical pattern-chasing, while the refinement explicitly identifies the recurrence that makes telescoping work.

LATE CHECKPOINT (STEP 240): SAME PROOF IDEA, MUCH SHORTER REFINEMENT

By step 240 the ThinkTwice reasoning trace has learned the right proof family: it now factors correctly and even states the recurrence

$$a_k = b_{k+1}$$

for the telescoping terms. But the base answer is still long and exploratory, with repeated recalculations and explicit small-n checks. A representative moment is:

ThinkTwice reasoning, step 240 (correct; 1222 words)

"Wait, let's do it carefully." Later, after reaching the closed form, it still spends another long block numerically checking $n = 1$ and $n = 2$ from scratch.

The refinement compresses that same proof into a direct b_{k+1}/b_k argument:

ThinkTwice refinement, step 240 (correct; 358 words)

It defines

$$b_k = 8k^2 - 12k + 5,$$

computes

$$b_{k+1} = 8k^2 + 4k + 1,$$

and then writes

$$\prod_{k=1}^{n} \frac{b_{k+1}}{b_k} = \frac{b_{n+1}}{b_1} = 8n^2 + 4n + 1.$$

The answer is then immediately boxed as $\boxed{8n^2 + 4n + 1}$.

The same-step pure-GRPO baseline remains less stable. Its base answer is still wrong, and even its correct refinement is over three times longer than ThinkTwice's and continues to rely on guess-and-check language:

Pure-GRPO refinement, step 240 (correct; 1067 words)

After deriving the same single-term ratio, it still pauses for several empirical checks: "Wait: $E_1 = 13$ ... $E_2 = 41$ ... $E_3 = 85$? Let's verify." Only later does it box $8n^2 + 4n + 1$.

This case is the cleanest qualitative support for the "explore early, exploit later" story. Early on, refinement discovers the missing structure that the base answer cannot yet use. Later on, once the base has learned the right route, refinement becomes the short and decisive version of that same route.

D.2 Case B: sample 497 — completing a stalled geometric derivation, then normalizing the final answer

Problem and role in the appendix

Problem (MATH500, level 5, sample 497). An equiangular octagon has four sides of length 1 and four sides of length $\frac{\sqrt{2}}{2}$, arranged so that no two consecutive sides have the same length. What is the area of the octagon?

Ground truth: $\boxed{\frac{7}{2}}$.

Why this case is useful. This is our clearest "finish the job" geometry example. At step 110 the ThinkTwice reasoning trace has the right global setup — directions, vectors, cumulative coordinates, and shoelace — but it stalls in the area computation and never produces a final boxed answer. The refinement keeps the same coordinate plan and makes it executable. By step 250 the base trace is correct, but it is still long and reports the area as the decimal 3.5; the refinement keeps the same proof while shortening it and normalizing the answer to the exact form $\frac{7}{2}$.

Checkpoint	Run	Base output	refinement output
Step 110	ThinkTwice	WRONG; 1280 words; no final boxed answer	CORRECT; 633 words; $\boxed{\frac{7}{2}}$
	Pure GRPO	WRONG; 1284 words; no final boxed answer	WRONG; 970 words; no final boxed answer
Step 250	ThinkTwice	CORRECT; 1303 words; $\boxed{3.5}$	CORRECT; 591 words; $\boxed{\frac{7}{2}}$
	Pure GRPO	CORRECT; 1209 words; $\boxed{\frac{7}{2}}$	CORRECT; 884 words; $\boxed{\frac{7}{2}}$

Table 9: Case B summary. Early refinement completes the coordinate-shoelace derivation; late refinement keeps the method but presents it more cleanly.

Early checkpoint (step 110): the base has the right plan but cannot close it

The ThinkTwice reasoning answer chooses a sensible geometry strategy: it sets the side directions to $0°, 45°, \ldots, 315°$, alternates the side lengths, constructs the cumulative coordinates, and then moves to the shoelace formula. The failure is not conceptual; it is procedural. The trace keeps changing how it wants to compute the area:

ThinkTwice reasoning, step 110 (wrong; 1280 words)

The trace reaches the area stage and then hesitates: "We can compute the area of a polygon given its vertices using the shoelace formula. Wait — actually, that's not quite right. Better: use the vector cross-product method." It never reaches a final boxed answer.

The refinement keeps the same coordinate walk but turns it into a checkable proof. It explicitly verifies the alternating side assignment, the vector directions, the cumulative coordinates, and finally the shoelace sum:

ThinkTwice refinement, step 110 (correct; 633 words)

"Side 1: length 1, $0° \to (1, 0)$ ... Side 8: length $\frac{\sqrt{2}}{2}$, $315° \to (\frac{1}{2}, -\frac{1}{2})$." After listing the vertices, it computes the shoelace terms and concludes with

$$\boxed{\frac{7}{2}}.$$

The same-step pure-GRPO baseline is informative because it fails in exactly the place where ThinkTwice's refinement succeeds. The baseline refinement is still expanding shoelace terms at the output limit and never boxes a final answer:

Pure-GRPO refinement, step 110 (wrong; 970 words)

The last available lines are still inside the term-by-term expansion,

$$x_6y_7 - x_7y_6 = \left(-\frac{\sqrt{2}}{2}\right)\left(\frac{\sqrt{2}}{2}\right) - \left(-\frac{\sqrt{2}}{2}\right)\left(1 + \frac{\sqrt{2}}{2}\right),$$

with no final boxed answer.

This is a useful mechanism example because refinement is not inventing a completely different strategy. The base trace already knows the right global method; refinement makes it *executable*, removes the indecision at the area step, and finishes the derivation.

LATE CHECKPOINT (STEP 250): SAME GEOMETRY, CLEANER ENDING, EXACT FINAL FORM

By step 250 the ThinkTwice reasoning answer solves the problem, but the trace is still long and numerically heavy. It ends with the decimal result

$$\boxed{3.5},$$

after a lengthy shoelace calculation. The last few lines read like a running calculator tape:

ThinkTwice reasoning, step 250 (correct; 1303 words)

"$0 + 0.5 + 1.5 + 1.5 + 2.0 + 1.0 + 0.5 + 0 = 7.0$. So total sum = 7.0. Area = $\frac{1}{2} \times |7.0| = \boxed{3.5}$."

The refinement keeps the same coordinate proof but tightens the presentation and restores the exact answer format:

ThinkTwice refinement, step 250 (correct; 591 words)

It verifies each side vector once, summarizes the shoelace sum compactly, and then states

$$\boxed{\frac{7}{2}}.$$

The refinement keeps the proof exact instead of finishing in decimal form.

The pure-GRPO baseline is also correct at this late checkpoint, but its refinement is still substantially longer (884 words versus 591) and includes an extra "reasonableness check" section after the exact shoelace computation. This is the kind of late-stage difference that is easy to miss if one looks only at accuracy: both systems are correct, but ThinkTwice's refinement is much closer to the polished proof style we would actually want to show a user.

D.3 CASE C: SAMPLE 194 — CONCRETE ERROR REPAIR EARLY, CONCISE PROOF LATE

Problem and role in the appendix

Problem (MATH500, level 5, sample 194). Find the maximum value of

$$\frac{x-y}{x^4+y^4+6}$$

over all real numbers x and y.

Ground truth: $\boxed{\frac{1}{4}}$.

Why this case is useful. This case separates two different refinement behaviors. At step 30 the base trace reaches the right one-variable reduction but makes a concrete derivative arithmetic error; the refinement explicitly identifies and fixes that error. By step 220 the base trace is correct, yet still wanders through multiple abandoned ideas before it settles on the final proof. The refinement turns that wandering derivation into a much shorter argument. The same-step pure-GRPO baseline is already correct at both checkpoints, so this case is *not* about a baseline failure; it is included because it shows that ThinkTwice learns both concrete bug repair and later proof compression on the same problem.

Checkpoint	Run	Base output	refinement output
Step 30	ThinkTwice	WRONG; 1204 words; no final boxed answer	CORRECT; 719 words; $\boxed{\frac{1}{4}}$
	Pure GRPO	CORRECT; 1236 words; $\boxed{\frac{1}{4}}$	CORRECT; 1181 words; $\boxed{\frac{1}{4}}$
Step 220	ThinkTwice	CORRECT; 1373 words; $\boxed{\frac{1}{4}}$	CORRECT; 374 words; $\boxed{\frac{1}{4}}$
	Pure GRPO	CORRECT; 906 words; $\boxed{\frac{1}{4}}$	CORRECT; 755 words; $\boxed{\frac{1}{4}}$

Table 10: Case C summary. Early refinement fixes a concrete derivative mistake; late refinement turns a wandering proof into a short direct argument.

EARLY CHECKPOINT (STEP 30): EXPLICIT DERIVATIVE BUG REPAIR

The ThinkTwice reasoning answer makes a strong start. It correctly reduces the problem to maximizing

$$f(d)=\frac{d}{\frac{1}{8}d^4+6}$$

by minimizing x^4+y^4 at fixed difference $d=x-y$. The failure is a concrete arithmetic error in the derivative step:

ThinkTwice reasoning, step 30 (wrong; 1204 words)

After deriving

$$f'(d)=\frac{-\frac{3}{8}d^4+6}{(\frac{1}{8}d^4+6)^2},$$

it sets the numerator to zero and writes

$$-\tfrac{3}{8}d^4+6=0\Rightarrow d^4=1,$$

which is wrong; the correct value is $d^4=16$.

The refinement is unusually explicit about what went wrong:

ThinkTwice refinement, step 30 (correct; 719 words)

"Mistake in previous step: I incorrectly computed $d^4 = 1$ instead of $d^4 = 16$." It then corrects the derivative equation to

$$-\tfrac{3}{8}d^4 + 6 = 0 \Rightarrow d^4 = 16 \Rightarrow d = 2,$$

substitutes $x = 1$, $y = -1$, and concludes with $\boxed{\frac{1}{4}}$.

The pure-GRPO baseline already solves this problem at the same early checkpoint, so the point of this example is narrower: ThinkTwice's refinement learns a genuine *bug-fixing* behavior, not just a generic "rewrite more clearly" behavior.

LATE CHECKPOINT (STEP 220): PRESERVING CORRECTNESS WHILE REMOVING EXPLORATORY CLUTTER

At step 220 the ThinkTwice reasoning answer is correct, but it is still visibly exploratory. It tries multiple proof routes, including one that it explicitly rejects:

ThinkTwice reasoning, step 220 (correct; 1373 words)

The trace first tries an inequality route,

$$z^4 + 1 \geq 4z \quad \text{for all real } z,$$

then immediately checks it at $z = 1$ and notes "$1 - 4 + 1 = -2$ — not non-negative, so that inequality fails." Only after this detour does it return to the correct calculus proof.

The refinement starts much closer to the final proof. It opens with the extremal point, then proves optimality directly:

ThinkTwice refinement, step 220 (correct; 374 words)

"We try a substitution: let $x = 1$, $y = -1$." It then proves

$$\frac{x - y}{x^4 + y^4 + 6} \leq \frac{1}{4}$$

by minimizing

$$f(x, y) = x^4 + y^4 - 4x + 4y + 6,$$

checks the critical point $(1, -1)$, and ends with $\boxed{\frac{1}{4}}$.

The same-step pure-GRPO baseline is also correct, but its refinement is more exploratory and almost twice as long. It begins with several test points (e.g., $x = 1, y = 0$; $x = 1, y = -1$; $x = 2, y = -1$) before moving to the formal proof. This makes the late-stage contrast very clear: once the base policy has learned the right answer, ThinkTwice's refinement behaves like an exploitation layer that strips away the dead ends.

D.4 CASE D: SAMPLE 425 — VECTOR RECOMPUTATION AS A BACKUP GEOMETRY CASE

Problem and role in the appendix

Problem (MATH500, level 5, sample 425). Let G and H denote the centroid and orthocenter of triangle ABC, respectively. Let F be the midpoint of $\overline{GH}$. Express $AF^2 + BF^2 + CF^2$ in terms of the side lengths a, b, c and circumradius R.

Ground truth: $\boxed{3R^2}$.

Why this case is useful. This is a strong backup geometry example. Early on, the ThinkTwice reasoning trace carries out the right vector setup but leaves the derivation in an unresolved mixed term involving $\vec{S} \cdot \vec{O}$. The refinement recomputes the vector expansion carefully and substitutes the missing circumcenter identity. By step 220 the base trace has learned the cleaner circumcenter-at-origin derivation, and the refinement compresses it further.

Checkpoint	Run	Base output	refinement output
Step 100	ThinkTwice	WRONG; 1138 words; no final boxed answer	CORRECT; 795 words; $\boxed{3R^2}$
	Pure GRPO	CORRECT; 711 words; $\boxed{3R^2}$	CORRECT; 1046 words; $\boxed{3R^2}$
Step 220	ThinkTwice	CORRECT; 746 words; $\boxed{3R^2}$	CORRECT; 410 words; $\boxed{3R^2}$
	Pure GRPO	CORRECT; 1259 words; $\boxed{3R^2}$	CORRECT; 993 words; $\boxed{3R^2}$

Table 11: Case D summary. A backup geometry example showing early recomputation and late compression.

EARLY CHECKPOINT (STEP 100): REFINEMENT REPAIRS THE VECTOR ALGEBRA

The ThinkTwice reasoning answer gets close to the goal but stops at an unresolved mixed term:

ThinkTwice reasoning, step 100 (wrong; 1138 words)

After expanding the squared norms, the base trace reduces the expression to

$$AF^2 + BF^2 + CF^2 = -2(\vec{S} \cdot \vec{O}) + 3\|\vec{O}\|^2 + \sum \|\vec{X}\|^2,$$

but never substitutes the identity needed to eliminate the remaining $\vec{S} \cdot \vec{O}$ term.

The refinement explicitly decides to recompute instead of trusting the previous algebra:

ThinkTwice refinement, step 100 (correct; 795 words)

"Let's recompute the expression carefully, term by term." It recomputes $\vec{S} \cdot \vec{H}$ and $\|\vec{H}\|^2$, then uses

$$\sum_{X=A,B,C} \left(\|\vec{X}\|^2 - 2\vec{X} \cdot \vec{O} + \|\vec{O}\|^2 \right) = 3R^2$$

to replace the mixed term and conclude with $\boxed{3R^2}$.

The pure-GRPO baseline is also correct here, but its refinement is substantially longer and contains its own visible false starts (including the line "Still wrong" while checking an auxiliary identity). This makes the ThinkTwice refinement look less like another exploratory search and more like a controlled repair.

LATE CHECKPOINT (STEP 220): CONCISE INVARIANT-BASED PROOF

By step 220 the ThinkTwice reasoning answer has already adopted the cleaner circumcenter-at-origin derivation. The refinement compresses it to a short invariant argument:

ThinkTwice refinement, step 220 (correct; 410 words)

It places the circumcenter at the origin, uses

$$\vec{H} = \vec{A} + \vec{B} + \vec{C}, \qquad \vec{G} = \tfrac{1}{3}(\vec{A} + \vec{B} + \vec{C}), \qquad \vec{F} = \tfrac{2}{3}(\vec{A} + \vec{B} + \vec{C}),$$

and then cancels the cross terms in

$$\sum AF^2 = 3\|\vec{F}\|^2 + \sum \|\vec{A}\|^2 - 2\vec{F} \cdot (\vec{A} + \vec{B} + \vec{C})$$

to get $\boxed{3R^2}$.

The pure-GRPO baseline is also correct at this checkpoint, but both its base and refinement are much longer (1259 and 993 words). This is a good backup example when we want a geometry case that is less about a wrong final answer and more about the shift from symbolic overhead to a clean invariant.

D.5 KEY TAKEAWAY FROM THE CASE STUDY

Across all four examples, the same qualitative pattern recurs. Early in training, refinement is most useful because it can *change what the model is doing*: it can discover a missing structural relation (Case A), finish a derivation that the base trace cannot close (Case B), or repair a concrete algebraic error (Case C). Later in training, once the base policy more often lands on the right route, refinement shifts toward *exploitation*: it preserves correctness while removing numerical detours, dead-end branches, and unnecessary verification blocks (Cases A–D). This is exactly the qualitative behavior we would expect if joint reasoning-and-refinement training is teaching a reusable refinement policy rather than merely adding another pass of unconstrained generation.