CURE: Critique-Driven Unified Reinforcement Learning for Test-Time Self-Improvement

Guirong Chen[1*], **Shuqi Ye**[1*], **Wenkai Yang**[1*†], **Shiqi Shen**[2],
Guangyao Shen[2], **Yankai Lin**[1‡]
[1]Gaoling School of Artificial Intelligence, Renmin University of China
[2]WeChat, Tencent Inc.
{aaaguirong, yankailin}@ruc.edu.cn

Abstract

The evolution paradigm of Large Language Models (LLMs) is shifting from scaling training compute to scaling inference-time compute. While Reinforcement Learning with Verifiable Rewards (RLVR) has become a key engine for this transition, standard approaches often fail to equip models with the autonomous improvement capabilities required for test-time scaling. Existing critique-guided methods attempt to mitigate this by leveraging external feedback or ground-truth signals; however, these dependencies are unavailable at test time, fundamentally limiting the model's capacity for continuous self-improvement. To bridge this gap, we propose **CURE** (**C**ritique-driven **U**nified **RE**inforcement Learning), a framework that jointly optimizes a single policy for standard solving, critiquing, and guided re-exploration. Uniquely, CURE facilitates re-exploration by generating strategic hints while discarding initial incorrect solutions to mitigate anchoring bias. Empirical results across diverse mathematical reasoning and code generation benchmarks demonstrate that CURE not only maintains competitive single-turn performance but, more importantly, unlocks effective inference-time scaling, enabling the model to significantly boost accuracy through iterative self-improvement.

1 Introduction

The evolution of Large Language Models (LLMs) (Ouyang et al., 2022; Yang et al., 2024) is witnessing a paradigm shift from scaling training-time compute to scaling test-time compute. Recent breakthroughs suggest that allocating more computational resources during inference can yield remarkable reasoning gains. This generally manifests in two paradigms: (1) *Parallel Scaling*, which improves performance by generating and evaluating multiple candidate solutions simultaneously (Wang et al., 2022; Snell et al., 2024); and (2) *Sequential Scaling*, which enhances reasoning by extending the chain of thought (CoT) to incorporate more deliberate reasoning behaviors (Yang et al., 2025d), typically induced through the Reinforcement Learning with Verifiable Rewards (RLVR) paradigm (Jaech et al., 2024; Guo et al., 2025).

However, current approaches to inference-time scaling remain relatively primitive and fall short of supporting human-like sequential refinement, characterized by iterative self-verification, critique, and correction. On one hand, parallel scaling merely samples independent solutions rather than iteratively improving them. On the other hand, current large reasoning models based on sequential scaling still struggle to perform reliable self-verification and correction (Kang et al., 2025), primarily because these capabilities are not explicitly optimized during training.

To address this, recent works have explored critique-guided exploration to improve the test-time improvement performance of LLMs, utilizing external feedback from stronger teachers (Xi et al., 2024; Yang et al., 2025b) or training specific critique models (Xi et al., 2025). Nevertheless, existing critique-guided methods suffer from fundamental dependencies that break the self-improvement loop during inference. They either rely on stronger external supervision that is impractical or prohibitively expensive, or depend on ground-truth signals to trigger the critique process—signals that are unavailable at test time. As a result, the policy model lacks the autonomy to solve, verify, and re-explore in a continuous loop, failing to realize the true potential of inference-time scaling (Zuo et al., 2025).

To bridge this gap, we propose **CURE** (**C**ritique-driven **U**nified **RE**inforcement Learning), a unified framework that enables models to explore and self-improve continuously at test time. As shown in

[*]Equal Contribution.
[†]Project Lead.
[‡]Corresponding Author.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 28632–28653
July 2-7, 2026

Figure 1, our framework jointly optimizes three atomic capabilities: (1) **Solving**: The fundamental ability to generate solutions for complex problems. (2) **Critiquing**: The capacity to verify the correctness of the model's own outputs (*Verification*), identify potential flaws and generate strategic hints for improvement (*Hint Generation*). (3) **Critique-Guided Re-Exploration**: The ability to leverage these self-generated hints to reconstruct reasoning paths. Through this unified training process, the model integrates these capabilities into a cohesive system, forming a self-contained improvement loop that enables robust inference-time scaling performance.

Specifically, our training process integrates these capabilities into a unified RLVR pipeline. A critical design choice in CURE is our handling of the re-exploration context. Unlike prior works (Zhang et al., 2025b; Xi et al., 2025) that append the entire history of incorrect attempts, we deliberately discard the initial incorrect solution during re-generation. Our pilot study (Figure 2) suggests that this design mitigates anchoring bias and prevents the model from repeating previous mistakes. Instead, we constrain the model to generate high-level strategic hints and force it to restart reasoning from a fresh state conditioned only on these hints. Finally, we optimize these capabilities within a unified objective: (1) Solving is enhanced by replaying successful re-exploration trajectories during RLVR; (2) Critiquing is optimized via direct supervised learning of golden verification judgments and RL with delayed accuracy-based rewards that prioritize hint utility; and (3) Re-exploration is aligned to effectively follow the generated strategic hints.

Extensive experiments on math and code tasks show that CURE achieves competitive single-turn performance and superior scalability. On AIME25, CURE already improves by 2.8% at the second turn and reaches a total gain of 4.2% through iterative self-improvement. These results suggest that CURE successfully instills a robust self-improvement mechanism into LLMs.

2 Related Work

RLVR for LLM Reasoning As a novel training paradigm for LLM reasoning, RLVR leverages verifiable reward functions, where they provide explicit binary feedback, e.g., assigning 1 to correct outputs and 0 to incorrect ones (Jaech et al., 2024; Guo et al., 2025; Team et al., 2025). Compared to reinforcement learning from human feedback (RLHF) (Ziegler et al., 2019; Ouyang et al., 2022), RLVR avoids reliance on subjective human evaluations or complex reward models, thereby making the training process more transparent and efficient. Consequently, RLVR is particularly well-suited to domains with crisp, executable evaluation protocols, such as mathematical reasoning (Hu et al., 2025; Zeng et al., 2025; He et al., 2025; Yang et al., 2025c) and code generation (Kumar et al., 2024; Xie et al., 2025), where solutions can be checked by symbolic solvers, compilers, or unit tests.

Reliable Critique for LLM Reasoning LLM-based critics offer a promising way to provide automated and scalable supervision for LLM outputs (Luo et al., 2023; McAleese et al., 2024; Ke et al., 2024). Existing approaches can be broadly categorized into two paradigms: (1) one line of work decouples the actor and critic, training a dedicated critique model to evaluate and comment on the actor's outputs (Xi et al., 2024, 2025; Yang et al., 2025b); (2) while another line unifies generation and critique within a single model, using self-generated feedback to drive refinement and learn how to revise responses accordingly (Zhang et al., 2025b; Li et al., 2025; Tang et al., 2025). In this work, we adopt the latter paradigm to equip a single model with unified capabilities for test-time self-improvement.

Guided Exploration for LLM Reasoning For complex and challenging tasks, prior works have shown that sampling a policy model multiple times may still fail to yield correct outcomes (Yue et al., 2025; Zhang et al., 2025b), motivating the need for more effective and guided exploration. For clarity, we categorize existing approaches into two groups. (1) The first group incorporates expert trajectories during training to shape priors over the search space (Yan et al., 2025; Huang et al., 2025; Ma et al., 2025; Fu et al., 2025; Zhang et al., 2025a), thus improving sample efficiency. However, these methods require costly, hard-to-scale expert supervision and may suffer from a distributional mismatch between expert demonstrations and the policy model's sampling distribution. (2) The second group guides exploration with external textual signals such as critiques or hints (Tang et al., 2025; Wang et al., 2025a; Yang et al., 2025b; Xi et al., 2025), e.g., by highlighting errors in previous responses to steer subsequent attempts. Our work builds on the second approach, but uses high-level

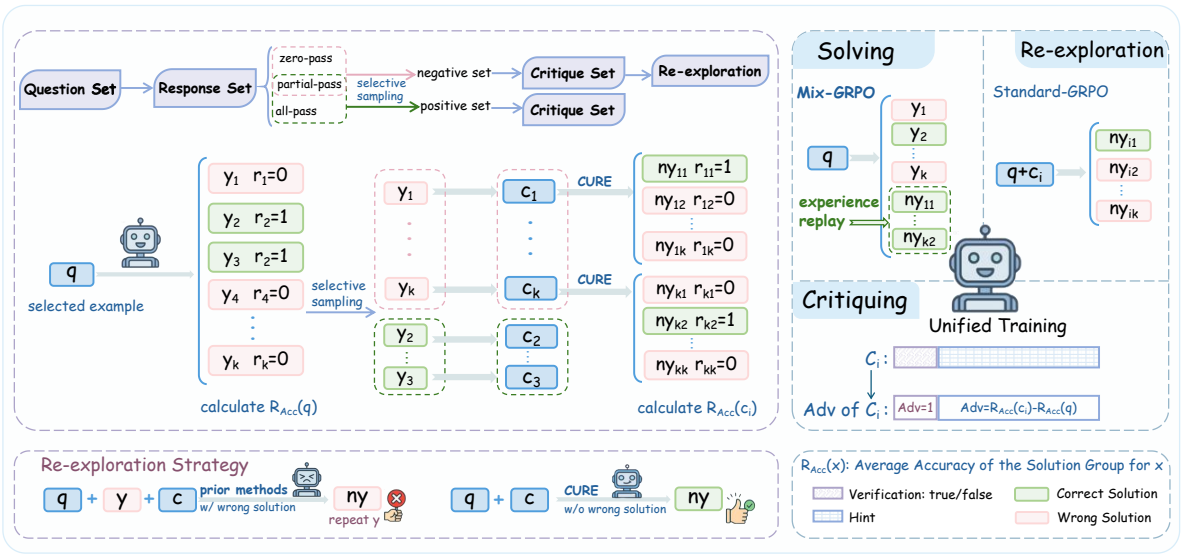


Figure 1: **Overview of our method CURE.** The left panel illustrates the overall pipeline and highlights the key differences between the baseline and CURE re-exploration strategy. The right panel shows a unified training framework that jointly optimizes multiple capabilities within a single policy.

hints without exposing incorrect solutions, in order to avoid anchoring bias induced by erroneous initial solutions and improve re-exploration.

3 CURE: Critique-Driven Unified Reinforcement Learning

Reinforcement Learning Objective We formulate the reasoning process as a Markov Decision Process (MDP), where a policy $\pi_{\boldsymbol{\theta}}$ generates a response $\boldsymbol{y} = (y_1, \ldots, y_T)$ given a query $\boldsymbol{x}$. The goal of RL is to maximize the expected reward $r(\boldsymbol{x}, \boldsymbol{y})$ provided by the environment, subject to a Kullback-Leibler (KL) divergence constraint to maintain training stability. The standard objective is defined as:

$$\begin{aligned}\mathcal{J}(\boldsymbol{\theta}) =& \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D},\boldsymbol{y}\sim\pi_{\boldsymbol{\theta}}}\Big[r(\boldsymbol{x},\boldsymbol{y}) \\ &- \beta\mathbb{D}_{\mathrm{KL}}\big(\pi_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x}) \,\|\, \pi_{\mathrm{ref}}(\cdot|\boldsymbol{x})\big)\Big],\end{aligned} \tag{1}$$

where β is the KL penalty coefficient and π_{ref} is a fixed reference policy (typically the initial model) used to prevent the tuned policy from deviating excessively from its original distribution.

Group Relative Policy Optimization (GRPO) GRPO (Shao et al., 2024) has been widely adopted in RLVR to enhance LLM reasoning. To optimize Eq. (1), policy gradient methods require estimating an advantage function A_t. GRPO leverages group-relative statistics to compute advantages, incurring low computational and memory overhead. Specifically, for each query $\boldsymbol{x}$, GRPO samples a group of G outputs $\{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_G\}$ from the old policy $\pi_{\boldsymbol{\theta}_{\mathrm{old}}}$. The advantage A_i for all tokens in the i-th output is derived by standardizing its reward against the group's distribution:

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

where r_i denotes the outcome-based reward. The PPO-style surrogate objective used in GRPO is formulated as:

$$\begin{aligned}\mathcal{J}_{\mathrm{GRPO}}(\boldsymbol{\theta}) =& \mathbb{E}_{\mathcal{D}}\Bigg[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|\boldsymbol{y}_i|}\sum_{t=1}^{|\boldsymbol{y}_i|}\mathrm{CLIP}\big(w_{i,t}(\boldsymbol{\theta}), A_i, \epsilon\big) \\ &- \beta\mathbb{D}_{\mathrm{KL}}\big(\pi_{\boldsymbol{\theta}} \,\|\, \pi_{\mathrm{ref}}\big)\Bigg],\end{aligned} \tag{3}$$

where $\mathrm{CLIP}(w, A, \epsilon) = \min(wA, \mathrm{clip}(w, 1-\epsilon, 1+\epsilon)A)$ limits the update step size to ensure training stability, and $w_{i,t}(\boldsymbol{\theta}) = \frac{\pi_{\boldsymbol{\theta}}(y_{i,t}|\boldsymbol{x}, \boldsymbol{y}_{i,<t})}{\pi_{\boldsymbol{\theta}_{\mathrm{old}}}(y_{i,t}|\boldsymbol{x}, \boldsymbol{y}_{i,<t})}$ represents the importance sampling ratio.

3.1 Framework Overview

As highlighted in Section 1, current critique-guided RLVR methods typically depend on external supervision or ground-truth signals to trigger critiques, which are either unavailable or impractical during inference. This dependency fundamentally restricts the model's ability to perform autonomous self-improvement at test time. To bridge this gap, we propose the **Critique-driven Unified REinforcement Learning (CURE)** framework.

Unlike prior approaches that treat critique as a mere auxiliary task for exploration, CURE is designed to integrate three critical capabilities—*Solving*, *Critiquing*, and *Critique-Guided Re-Exploration*—into a single, unified policy. The core intuition is that even when a model fails to solve a problem directly, it often possesses the latent capacity to recognize errors (Liu et al., 2025; Feng et al., 2025) and generate actionable hints. By explicitly optimizing the model to critique its own outputs and use those critiques to guide subsequent re-generation, we equip the model with an intrinsic self-improvement mechanism that functions independently of external feedback.

Our framework establishes a co-training process that achieves two objectives: (1) **Autonomous Self-Improvement Loop:** We train the model not only to solve problems but also to autonomously verify its answers and generate strategic hints. This removes the reliance on ground truths, enabling the self-improvement loop to operate seamlessly during inference. (2) **Unbiased Re-Exploration:** Distinct from methods that condition re-generation on incorrect initial solutions (Jiang et al., 2025; Zhang et al., 2025b), we propose to discard the faulty reasoning path during re-exploration, and leverage high-level strategic hints to guide exploration from a fresh context, avoiding anchoring bias.

Overview As illustrated in Figure 1, during each online RL iteration, the model first generates diverse solutions. We then selectively sample solution pairs to generate critiques containing verification judgments and strategic hints. For the guided re-generation, we concatenate the generated hint with the original problem—**explicitly discarding the incorrect initial solution**. Finally, the model is updated via a unified GRPO objective that jointly optimizes these capabilities.

3.2 Training Pipeline

Each CURE training iteration consists of three phases: Initial Solution Generation, Critique Generation, and Critique-Guided Re-Generation.

Phase 1: Initial Solution Generation For each problem $\boldsymbol{x}$, we first sample a group of K initial solutions $\mathcal{S}_{\text{init}} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_K\}$ from the current policy $\pi_{\boldsymbol{\theta}}$. These solutions are evaluated using an outcome verifier to obtain binary rewards $\{r_1, \ldots, r_K\}$, where $r_i \in \{0, 1\}$.

Phase 2: Critique Generation Rather than critiquing every response, we employ a *Selective Sampling Strategy* to construct a critique batch $\mathcal{B}_{\text{crit}}$ aimed at maximizing exploration efficiency while maintaining training stability. Specifically, we categorize queries based on their group-level accuracy ($\text{Acc}(\boldsymbol{x}) = \frac{1}{K}\sum_{i=1}^{K} r_i$) and prioritize sampling from "informative" instances—namely, *Zero-pass* ($\text{Acc} = 0$) and *Partial-pass* ($\text{Acc} \in (0, 1)$) groups—where the model either fails completely or exhibits inconsistency. Crucially, we enforce a balanced 1:1 ratio of positive samples (correct, $r_i = 1$) and negative samples (incorrect, $r_i = 0$) to prevent the policy from collapsing into trivial verification patterns, such as consistently predicting a single class. Once the batch is constructed, for each sampled pair $(\boldsymbol{x}, \boldsymbol{y}_i) \in \mathcal{B}_{\text{crit}}$, the model generates a critique $\boldsymbol{c}_i = (\boldsymbol{v}_i; \boldsymbol{h}_i)$. As detailed in Appendix A, this critique comprises a verification judgment $\boldsymbol{v}_i \in \{\text{Correct}, \text{Incorrect}\}$ and a strategic hint $\boldsymbol{h}_i$, which serves as a conceptual guide for the subsequent re-exploration.

Phase 3: Critique-Guided Re-Generation The strategic hint $\boldsymbol{h}_i$ serves as a scaffold for re-exploration. We construct a guided prompt by concatenating the original problem and the hint: $\boldsymbol{x}'_i = (\boldsymbol{x}; \boldsymbol{h}_i)$ (see Appendix A). A key distinction of CURE is the deliberate exclusion of the initial incorrect solution $\boldsymbol{y}_i$. Unlike prior self-correction paradigms that append the wrong reasoning trace to the context, we force the model to reconstruct the reasoning path based on the problem and high-level hint only. This design effectively mitigates the anchoring bias caused by initial errors. As supported by our pilot experiments in Figure 2, discarding the incorrect context can reduce the repetition of wrong answers ("SameRatio") and improve the overall re-generation accuracy compared to appending it (see Appendix C for details).

Conditioned on this fresh context, the model generates a group of M guided solutions $\mathcal{S}_{\text{guided}} = \{\boldsymbol{y}'_{i,1}, \ldots, \boldsymbol{y}'_{i,M}\}$. These solutions are evaluated against the ground truth to obtain binary outcome rewards $\{r'_{i,1}, \ldots, r'_{i,M}\}$. Crucially, we employ the outcome of this re-exploration to assign a *Delayed Reward* to the critique $\boldsymbol{c}_i$ itself. The total critique reward $R(\boldsymbol{c}_i)$ is defined as a combination of the guided generation accuracy and structural constraints:

$$R(\boldsymbol{c}_i) = \bar{r}'_i + R_{\text{format}}(\boldsymbol{c}_i).$$

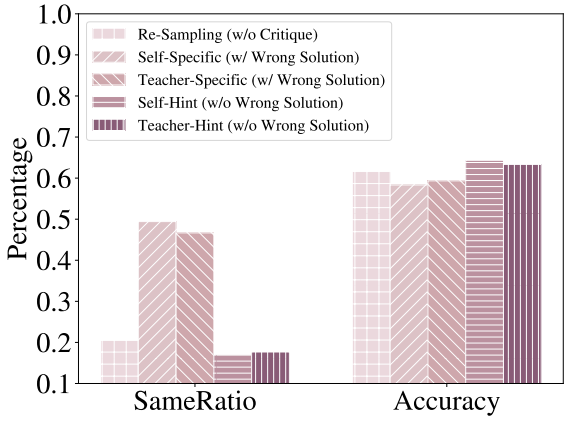


Figure 2: **Pilot Experiment Results.** SameRatio denotes the proportion of re-exploration attempts that overlap with the initial incorrect response, reflecting the model's tendency to repeat its previous errors. Accuracy is the percentage of problems solved correctly, either during the initial generation (Phase 1) or re-generation (Phase 3).

Here, $\bar{r}'_i = \frac{1}{M}\sum_{j=1}^{M} r'_{i,j}$ represents the average success rate of the re-exploration group, aligning the critique objective with the goal of reasoning: a valid critique is defined strictly by its utility in leading to a correct solution. The auxiliary term R_{format} ensures the structural validity of the critique by enforcing adherence to the predefined output schema (e.g., requiring specific keys like `"High_Level_Hint"`). We define the penalty function as:

$$R_{\text{format}}(\boldsymbol{c}_i) = \begin{cases} -\lambda_1 & \text{if required keys are missing,} \\ -\lambda_2 & \text{if hint length} < L_{\min}, \\ 0 & \text{otherwise,} \end{cases}$$

where λ_1 and λ_2 are hyperparameters penalizing format violations and trivial hints (e.g., length $<$ $L_{\min}$), respectively.

3.3 Unified Training Objective

Our framework optimizes a unified objective that jointly enhances the model's capabilities in standard solving, critiquing, and guided solving. Each task is formulated as a GRPO objective (Eq. (3)), differing only in input conditioning and task-specific advantage definitions. The total objective is defined as an additive combination:

$$\mathcal{J}_{\text{total}} = \mathcal{J}_{\text{solve}} + \mathcal{J}_{\text{critique}} + \mathcal{J}_{\text{guided}},$$

Solving Objective with Experience Replay The solving objective $\mathcal{J}_{\text{solve}}$ optimizes the standard generation policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})$. To effectively propagate the "0-to-1" breakthroughs achieved during re-exploration, we employ an *Experience Replay* strategy. Specifically, we selectively incorporate successful guided solutions $\mathcal{S}_{\text{success}} \subseteq \mathcal{S}_{\text{guided}}$ back into the initial solution group $\mathcal{S}_{\text{init}}$, treating them as if they were generated directly from the problem $\boldsymbol{x}$ (i.e., by masking the hint context).

The augmented group $\mathcal{S}_{\text{aug}} = \mathcal{S}_{\text{init}} \cup \mathcal{S}_{\text{success}}$ facilitates a critical re-calibration of advantages. The successful guided trajectories, serving as positive *anchors* with reward $r = 1$, significantly elevate the group mean reward, particularly for previously zero-pass groups. Under GRPO's normalization (Eq. (2)), this shift assigns strongly negative advantages to the original incorrect solutions while assigning positive advantages to the replayed ones. This mechanism provides the necessary gradient signal to disincentivize failure modes and reinforce correct reasoning paths.

Critique Objective with Hybrid Advantages The critique objective $\mathcal{J}_{\text{critique}}$ optimizes the critique generation policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{c}|\boldsymbol{x}, \boldsymbol{y})$. Given the composite structure of the critique—comprising a verification judgment $\boldsymbol{v}$ and a strategic hint $\boldsymbol{h}$—we design a *Hybrid Advantage Function* $A_{\text{crit}}(t)$ that adapts to the token segment:

$$A_{\text{crit}}(t) = \begin{cases} 1.0 & \text{if } t \in \text{Verification,} \\ R(\boldsymbol{c}_i) - b(\boldsymbol{x}) & \text{otherwise,} \end{cases}$$

where $b(\boldsymbol{x}) = \frac{1}{K}\sum_{i=1}^{K} r_i$ serves as the baseline, representing the average accuracy of the initial solution group.

For the *Verification Segment*, we assign a fixed positive advantage (1.0). In implementation, we explicitly prepend the ground-truth judgment (e.g., `Error_Found: true`, derived from the solution's correctness) to the beginning of the critique response $\boldsymbol{c}_i$. This effectively transforms the verification task into a supervised optimization (akin to behavior cloning), ensuring the model grounds its critique on correct judgments and improving learning efficiency.

For the subsequent *Hint Segment*, we utilize the centered advantage $R(\boldsymbol{c}_i) - b(\boldsymbol{x})$. This formulation provides a rigorous quality control mechanism: it yields positive advantage values only when the hint leads to a guided performance exceeding the model's baseline average ($R(\boldsymbol{c}_i) > b(\boldsymbol{x})$). Conversely, misleading or trivial hints that fail to improve the outcome receive negative or zero advantages. This contrastive signal effectively suppresses

the generation of unhelpful text, incentivizing the model to produce strategic, actionable guidance.

Guided Solving Objective Finally, the guided solving objective $\mathcal{J}_{\text{guided}}$ optimizes the policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{y}'|\boldsymbol{x}, \boldsymbol{h})$ to generate reasoning paths conditioned on the generated hints. This term ensures that the model remains compliant with its own strategic advice. Similar to the standard solving task, we apply the group-relative advantage normalization (Eq. (2)) within each guided solution group $\mathcal{S}_{\text{guided}}$ to facilitate stable policy updates.

Optimization Details. In practice, we implement the unified objective as follows. First, we compute advantages independently within each subtask group, which standardizes the optimization signals across groups and prevents any group from dominating training. Second, to handle variable output lengths, we average the GRPO loss of each training sample over its generated tokens, as reflected in Eq. (3). Finally, we form the overall objective by summing the three task losses with equal weights, using a fixed $1:1:1$ weighting scheme. This simple strategy works well for two reasons: (1) group-wise reward normalization together with per-token loss averaging naturally keeps the optimization scales of different samples comparable; and (2) the three sub-tasks use distinct prompt templates, which separate them in the input space and reduce destructive interference during joint optimization.

4 Experiments

4.1 Experimental Setup

Base Models and Baselines We conduct experiments on two target models: **Qwen2.5-7B-Instruct** (Yang et al., 2024) and **Qwen3-4B-Base** (Yang et al., 2025a). This selection covers both instruction-tuned and pre-trained models across different scales, allowing us to evaluate the generalization of CURE. We employ the GRPO (Shao et al., 2024) algorithm as our primary baseline.

Training Settings For mathematical reasoning, we construct our training dataset by downsampling a subset from DeepMath (He et al., 2025). To evaluate generalization, we also extend CURE to the code generation domain, following the setup in Wang et al. (2025b) to construct a training set from CodeContests (Li et al., 2022). All experiments are implemented using the `verl` framework (Sheng et al., 2025). For the GRPO baseline, we sample 8 responses per prompt. To ensure a fair comparison, we align the sampling budget for CURE by generating 4 initial solutions and 4 guided solutions per prompt, along with a single critique response for the selected batch. Complete hyperparameters and implementation details for both domains are provided in Appendix D.

Evaluation Settings We evaluate performance across two domains: (1) **Mathematics**: MATH500 (Hendrycks et al., 2020), AMC23, AIME24 (Zhang and Math-AI, 2024), AIME25 (Zhang and Math-AI, 2025), and OlympiadBench (He et al., 2024); (2) **Code Generation**: Codeforces (Penedo et al., 2025), MBPP (Austin et al., 2021), and LiveCodeBench (Jain et al., 2024). During inference, we set the sampling temperature to 0.6. We report two key metrics: (1) **Standard Single-turn Accuracy**: We report Avg@32 for smaller math datasets and Avg@2 for larger math and all code datasets to ensure reliability. (2) **Inference-time Self-Improvement**: This metric measures the final accuracy via our critique-guided loop. Specifically, we execute the re-generation process for up to 8 rounds (including the initial attempt), where the model re-generates only if its verification predicts "Incorrect". (see Appendix F.2 for a comparison with unconditional strategy). Detailed self-verification results are provided in Appendix F.1.

4.2 Main Results

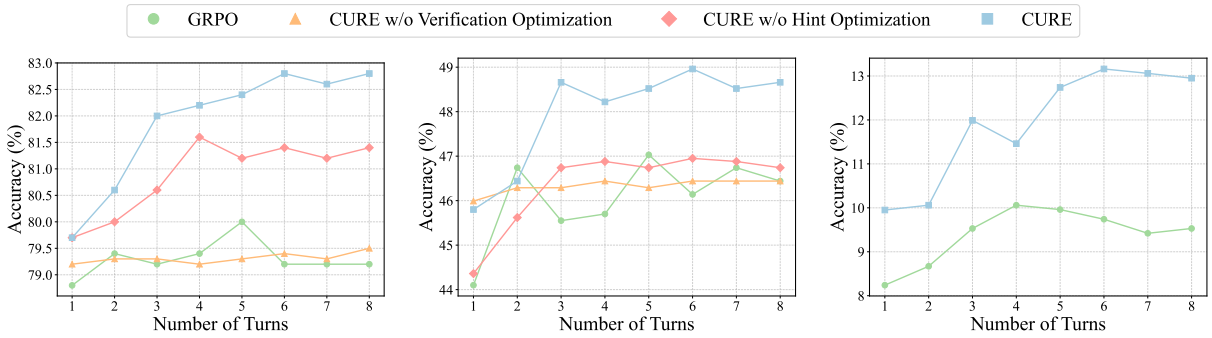
Table 1 summarizes the performance of CURE across mathematical reasoning and code generation benchmarks.

In the standard single-turn setting, CURE consistently maintains comparable or superior performance relative to the GRPO baseline. For example, on Qwen2.5-7B-Instruct, CURE achieves an average accuracy improvement of **+1.0%** on math tasks and competitive performance gains on coding tasks (e.g., **+1.8%** on CodeForces). We attribute these gains primarily to the *Experience Replay* mechanism. By selectively replaying successful re-generated solutions for previously zero-pass groups, CURE effectively injects positive learning signals into the sparse reward landscape of hard problem clusters, thereby transforming zero-gradient stagnation into effective learning.

The primary contribution of CURE lies in its ability to unlock effective test-time self-improvement. A central hypothesis of this work

Method	Mathematical Reasoning Accuracy						Code Generation Accuracy			
	AIME24	AIME25	AMC	MATH	Olym.	Avg.	MBPP	LCB	CF	Avg.
Qwen2.5-7B-Instruct										
Base Model	11.3	8.9	52.2	74.9	39.4	37.3	64.0	25.4	5.8	31.7
GRPO	14.2	10.6	57.5	78.8	44.1	41.0	83.5	29.1	8.2	40.3
+ *Self-Improve (T=8)*	13.3	10.5	57.7	79.2	46.4	41.4	83.2	**30.6**	9.5	41.1
CURE (Ours)	14.1	11.8	58.4	79.7	45.8	42.0	84.3	27.3	10.0	40.5
+ *Self-Improve (T=8)*	**16.0**	**16.0**	**60.9**	**82.8**	**48.7**	**44.9**	**85.6**	28.5	**13.0**	**42.4**
Qwen3-4B-Base										
Base Model	9.4	5.9	40.2	67.3	35.8	31.7	62.4	6.3	1.8	23.5
GRPO	20.9	17.5	65.7	**85.3**	51.8	48.2	78.3	19.2	12.4	36.6
+ *Self-Improve (T=8)*	20.8	18.3	66.7	85.2	53.2	48.8	76.3	18.7	12.8	35.9
CURE (Ours)	21.4	17.3	66.8	84.5	52.3	48.5	79.4	23.5	12.5	38.5
+ *Self-Improve (T=8)*	**23.5**	**19.7**	**70.1**	85.1	**54.1**	**50.5**	**82.1**	**25.9**	**13.1**	**40.4**

Table 1: Main results on mathematical reasoning and code generation benchmarks. We report the accuracy for standard generation and the performance with inference-time self-improvement (up to $T = 8$ rounds). **Olym.**: OlympiadBench; **LCB**: LiveCodeBench; **CF**: CodeForces.



(a) Results on **MATH500**. (b) Results on **OlympiadBench**. (c) Results on **CodeForces**.

Figure 3: **Inference-time self-improvement results** on mathematical (a, b) and coding (c) benchmarks using Qwen2.5-7B-Instruct. The plots track accuracy changes over $T = 8$ iterative rounds. For the coding task, results are reported only for GRPO and CURE, as ablation variants are primarily analyzed within the mathematical domain.

is that CURE instills a dynamic self-improvement mechanism that can be leveraged during inference.

As shown in the "+ *Self-Improve*" rows of Table 1, CURE demonstrates a significant and sustained performance boost through iterative self-improvement. On Qwen2.5-7B-Instruct, inference-time scaling elevates the average accuracy from 42.0% to **44.9%**, significantly surpassing the baseline's scaling ceiling of 41.4%. Notably, this advantage is domain-agnostic: on CodeForces, scaling with CURE yields a substantial **+3.0%** improvement over its single-turn baseline, whereas GRPO shows minimal gain (+1.3%). These results confirm that CURE instills a robust policy that can perform effective test-time self-improvement across diverse reasoning domains.

Question:
The area of triangle ABC is 6 square centimeters. $AB \parallel DE$, $BD = 4BC$.
What is the number of square centimeters in the area of riangle CDE?
draw ((-.3,-3)--(.1,1)--(-1,0)--(3,0)--cycle); A(.1,1);B,(-1,0);C(0,0); D(3,0); E(-.3,-3)

Turn1:
Solution1:... We should have $BD = 4BC$ implying that point D is outside of BC extended....Let $BC=x$, then $BD = 4x$, and the total length $CD = BD + BC = 4x + x = 5x$... Answer=150
Hint1: Always double-check the geometric relationships and segment placements to ensure they are consistent with the given conditions.

Turn4:
Solution4: ...Since D lies on AC, we can express the length AD as:$AD = AC - CD$ .... Answer:frac{6}{25}
Hint4: Always double-check the given conditions and how they relate to the geometric configuration to ensure the correct setup of ratios and proportions.

Turn5:
Solution5: To solve for the area of triangle CDE let's carefully analyze...Let $BC = x$, then $BD = 4x$. Therefore, $CD = BD - BC = 4x - x = 3x$.... Answer=54

Figure 4: Multi-turn self-improvement case study.

The trajectory analysis in Figure 3 further reveals the qualitative difference between methods. CURE extends the effective scaling window to 5-6 rounds, achieving a higher and more stable peak accuracy. In contrast, the GRPO baseline exhibits only a marginal initial rise followed by stagnation

Inference Strategy	Budget	GRPO	CURE
Standard (Pass@1)	1	78.8	79.7
Majority Voting	4	81.4	82.2
	8	81.8	82.6
Best-of-N	4	**82.6**	81.2
(Math-Shepherd-7B)	8	**82.6**	81.0
Iterative Self-Improve	4	79.4	82.4
(Ours)	8	79.2	**82.8**

Table 2: Comparison of inference-time strategies on MATH with Qwen2.5-7B-Instruct (%). For majority voting and Best-of-N, the budget is the number of sampled candidates N; for iterative self-improvement, it is the maximum number of refinement turns T.

Method	Acc. (%)	Avg. Tokens	Latency (s)
GRPO + BoN (N=8)	82.6	∼6400	∼0.44
CURE + Iter. (T=8)	**82.8**	∼1827	∼**0.43**

Table 3: Efficiency and latency trade-off on MATH500 under the same maximum generation budget. Avg. Tokens denotes the total number of generated tokens per query. For BoN, we conservatively exclude the additional latency of the external reward model used for reranking.

or oscillation. This disparity highlights that standard RLVR fails to equip the model with a constructive self-improvement mechanism. By contrast, our unified training framework successfully converts test-time compute into tangible performance gains.

To further inspect the self-improvement process, we present a concrete case study in Figure 4. As shown, from Turn 1 to Turn 4, the model consistently misinterprets the geometric configuration across different aspects. Guided by self-generated hints, the model is prompted to re-evaluate the relationships among segments, ultimately arriving at the correct solution in Turn 5.

4.3 Inference-Time Scaling Analysis

To further analyze how CURE uses additional test-time budget, we compare several inference strategies on MATH500 with Qwen2.5-7B-Instruct under the same maximum generation budget. Specifically, we consider standard decoding (Pass@1), majority voting, Best-of-N with an external reward model, and our iterative self-improvement. Here, N denotes the number of parallel samples for majority voting and Best-of-N, while T denotes the maximum number of sequential refinement turns for iterative self-improvement. Since these strategies allocate the generation budget differently (parallel sampling versus sequential refinement), we further report token and latency statistics in Table 3. For Best-of-N, we use Math-Shepherd-7B as the external reward model for candidate selection.

As shown in Table 2, CURE consistently outperforms GRPO under iterative self-improvement and majority voting strategies. Notably, under the Best-of-N strategy, CURE's performance slightly drops and underperforms GRPO. We hypothesize this is primarily due to a distribution shift: the external reward model (Math-Shepherd-7B) is largely trained on standard, straightforward reasoning trajectories. Consequently, it may struggle to accurately evaluate—or might actively penalize—the unique self-correction patterns, reflection steps, or revised trajectories inherently generated by CURE. This limitation of relying on an out-of-distribution external reward model further highlights the advantage of our Iterative Self-Improvement approach. By relying on the model's intrinsic ability to refine its own outputs, our strategy avoids the format-alignment issues of external rerankers and achieves the highest overall performance.

To further quantify the practical trade-off between accuracy and deployment cost, we compare GRPO + Best-of-N ($N = 8$) against CURE + iterative self-improvement ($T = 8$) in terms of average generated tokens and latency per query on MATH500. As shown in Table 3, CURE achieves slightly higher accuracy while using substantially fewer generated tokens, with comparable reported latency in this setup. This indicates that CURE converts additional inference-time computation into more efficient self-improvement rather than merely increasing parallel sampling. We further provide the corresponding efficiency–latency comparison on OlympiadBench in Appendix F.4. On this harder benchmark, CURE still uses fewer generated tokens, but its sequential refinement loop incurs higher latency, highlighting that the latency advantage is task-dependent rather than universal.

4.4 Analysis of Exploration Dynamics

To evaluate CURE's ability to overcome exploration stagnation, we visualize the training dynamics of *zero-pass problems*—queries for which no correct solutions are obtained across all attempts—in Figure 5. For such samples, standard GRPO receives no effective learning signal.

Although CURE initially shows a higher failure rate due to unstable critiques, it exhibits a crossover

and sustained decline after about 200 steps, eventually dropping significantly below the GRPO baseline. This trajectory confirms that once the critique capability is grounded and continuously optimized, the guided re-exploration mechanism effectively converts hard failures into successes, unlocking effective learning signals for these hard samples. Furthermore, the ablation variant that does not optimize the hint generation ability (*w/o Hint Optimization*) plateaus at a higher error rate than CURE, underscoring the necessity of explicit hint optimization for sustaining long-term exploration gains.

4.5 Ablation Studies

To disentangle the contributions of key components in our CURE framework, we conduct a series of ablation studies. We first analyze the two sub-components of the Critique—Verification and Hint—by selectively removing their optimization objectives. Subsequently, we examine the impact of the Experience Replay strategy.

Impact of Verification Optimization We first examine the variant w/o Verification Optimization, where the advantage for the verification segment is set to 0 (i.e., not trained). As shown in Figure 3, this model maintains a similar single-turn accuracy to the full CURE, suggesting that verification supervision does not significantly affect the base solving capability. However, its scaling trajectory is nearly flat. Lacking a calibrated verification ability, the model cannot effectively distinguish between correct and incorrect solutions during inference, rendering the conditional re-generation strategy ineffective. This underscores that a robust self-verification capability is crucial for successful inference-time self-improvement.

Impact of Hint Optimization Next, we evaluate the variant w/o Hint Optimization, where the hint generation is not explicitly optimized via RL (the verification is still trained). We observe two key phenomena: (1) **Lower Initial Performance**: Particularly on harder benchmarks like OlympiadBench, the starting accuracy is noticeably lower than CURE. This confirms that without optimizing the hint generation policy, the quality of feedback during training is suboptimal, limiting the model's ability to explore complex solution spaces. (2) **Limited Scaling Potential**: Although this variant shows some initial improvement in the first 3-4 rounds, it peaks earlier and achieves a lower total gain compared to the full CURE. This indicates that explicitly optimizing hint generation is essential for sustaining long-term growth during inference-time scaling.

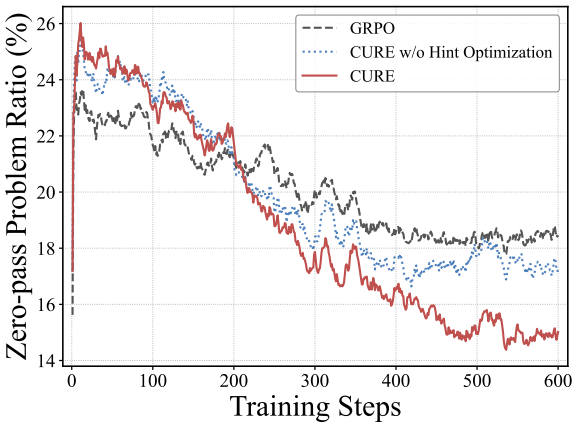


Figure 5: **Dynamics of the zero-pass problem ratio during training** (on Qwen2.5-7B-Instruct, smoothed via EMA).

Impact of Experience Replay Strategy Finally, we investigate whether to incorporate *all* successful guided solutions into corresponding solution groups or replaying only those from zero-pass groups. Our empirical results indicate that prioritizing the latter yields better training stability and final performance. We put the detailed results and comparisons in Appendix F.6.

5 Conclusion

In this work, we introduce **CURE**, a unified RL framework that empowers LLMs with autonomous self-improvement capabilities. By jointly optimizing solving, critiquing, and re-exploration within a single policy, CURE eliminates dependencies on external supervision during test-time exploration. Our experiments in math and code domains demonstrate that CURE not only maintains competitive single-turn performance but also unlocks effective inference-time scaling potential, enabling models to continuously perform a *solve–verify–critique–explore* loop. Future work may extend CURE to open-ended domains and scale the iterative loop to more complex, multi-turn reasoning chains.

Limitations

Though effective in enhancing the test-time self-improvement abilities of LLMs, our work has some limitations: (1) We conduct experiments only in verifiable domains (i.e., mathematics and code),

where deterministic reward signals can be naturally leveraged to optimize each capability within our framework. Extending our method to open-domain tasks, where supervision is less explicit and rewards are inherently noisy, remains a practical yet challenging direction for future work. (2) Compared with vanilla RLVR, our framework incurs higher computational costs at each optimization step due to the joint optimization of multiple model capabilities. However, these additional costs enable the model to achieve sustained performance gains at test time through continuous self-improvement, outperforming vanilla baselines.

Ethics Statement

In this work, we introduce a unified training framework that endows a single policy model with standard solving, self-critiquing, and critique-guided re-exploration abilities. Our method enables the optimized model to achieve continual test-time self-improvement via an iterative loop of proposing, verifying, critiquing, and re-exploring new solutions. Our work makes a positive step toward effective self-evolution of LLMs.

Acknowledgments

We sincerely thank all the anonymous reviewers and AC(s) for their helpful feedback and constructive suggestions. This work was supported by The National Natural Science Foundation of China (No. 62376273) and Beijing Natural Science Foundation (L253001).

References

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Zhangying Feng, Qianglong Chen, Ning Lu, Yongqian Li, Siqi Cheng, Shuangmu Peng, Duyu Tang, Shengcai Liu, and Zhirui Zhang. 2025. Is prm necessary? problem-solving rl implicitly induces prm capability in llms. *arXiv preprint arXiv:2505.11227*.

Yuqian Fu, Tinghong Chen, Jiajun Chai, Xihuai Wang, Songjun Tu, Guojun Yin, Wei Lin, Qichao Zhang, Yuanheng Zhu, and Dongbin Zhao. 2025. Srft: A single-stage method with supervised and reinforcement fine-tuning for reasoning. *arXiv preprint arXiv:2506.19767*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, and 1 others. 2024. 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)*, pages 3828–3850.

Zhiwei He, Tian Liang, Jiahao Xu, Qiuzhi Liu, Xingyu Chen, Yue Wang, Linfeng Song, Dian Yu, Zhenwen Liang, Wenxuan Wang, and 1 others. 2025. Deepmath-103k: A large-scale, challenging, decontaminated, and verifiable mathematical dataset for advancing reasoning. *arXiv preprint arXiv:2504.11456*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

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

Hsiu-Yuan Huang, Chenming Tang, Weijie Liu, Saiyong Yang, and Yunfang Wu. 2025. Think outside the policy: In-context steered policy optimization. *arXiv preprint arXiv:2510.26519*.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, and 1 others. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Livecodebench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*.

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

Liwei Kang, Yue Deng, Yao Xiao, Zhanfeng Mo, Wee Sun Lee, and Lidong Bing. 2025. First try matters: Revisiting the role of reflection in reasoning models. *arXiv preprint arXiv:2510.08308*.

Pei Ke, Bosi Wen, Andrew Feng, Xiao Liu, Xuanyu Lei, Jiale Cheng, Shengyuan Wang, Aohan Zeng,

Yuxiao Dong, Hongning Wang, and 1 others. 2024. Critiquellm: Towards an informative critique generation model for evaluation of large language model generation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13034–13054.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, and 1 others. 2024. Training language models to self-correct via reinforcement learning. *arXiv preprint arXiv:2409.12917*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626.

Ang Li, Yifei Wang, Zhihang Yuan, Stefanie Jegelka, and Yisen Wang. 2025. Lanpo: Bootstrapping language and numerical feedback for reinforcement learning in llms. *arXiv preprint arXiv:2510.16552*.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, and 1 others. 2022. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097.

Xiaoyuan Liu, Tian Liang, Zhiwei He, Jiahao Xu, Wenxuan Wang, Pinjia He, Zhaopeng Tu, Haitao Mi, and Dong Yu. 2025. Trust, but verify: A self-verification approach to reinforcement learning with verifiable rewards. *arXiv preprint arXiv:2505.13445*.

Liangchen Luo, Zi Lin, Yinxiao Liu, Lei Shu, Yun Zhu, Jingbo Shang, and Lei Meng. 2023. Critique ability of large language models. *arXiv preprint arXiv:2310.04815*.

Lu Ma, Hao Liang, Meiyi Qiang, Lexiang Tang, Xiaochen Ma, Zhen Hao Wong, Junbo Niu, Chengyu Shen, Runming He, Yanhao Li, and 1 others. 2025. Learning what reinforcement learning can't: Interleaved online fine-tuning for hardest questions. *arXiv preprint arXiv:2506.07527*.

Nat McAleese, Rai Michael Pokorny, Juan Felipe Ceron Uribe, Evgenia Nitishinskaya, Maja Trebacz, and Jan Leike. 2024. Llm critics help catch llm bugs. *arXiv preprint arXiv:2407.00215*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, and 1 others. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Guilherme Penedo, Anton Lozhkov, Hynek Kydlíček, Loubna Ben Allal, Edward Beeching, Agustín Piqueres Lajarín, Quentin Gallouédec, Nathan Habib, Lewis Tunstall, and Leandro von Werra. 2025. Codeforces. https://huggingface.co/datasets/open-r1/codeforces.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 1279–1297.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*.

Chenming Tang, Hsiu-Yuan Huang, Weijie Liu, Saiyong Yang, and Yunfang Wu. 2025. Do not step into the same river twice: Learning to reason from trial and error. *arXiv preprint arXiv:2510.26109*.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, and 1 others. 2025. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*.

Xinyi Wang, Jinyi Han, Zishang Jiang, Tingyun Li, Jiaqing Liang, Sihang Jiang, Zhaoqian Dai, Shuguang Ma, Fei Yu, and Yanghua Xiao. 2025a. Hint: Helping ineffective rollouts navigate towards effectiveness. *arXiv preprint arXiv:2510.09388*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Yinjie Wang, Ling Yang, Ye Tian, Ke Shen, and Mengdi Wang. 2025b. Co-evolving llm coder and unit tester via reinforcement learning. *arXiv preprint arXiv:2506.03136*.

Zengzhi Wang, Fan Zhou, Xuefeng Li, and Pengfei Liu. 2025c. Octothinker: Mid-training incentivizes reinforcement learning scaling. *arXiv preprint arXiv:2506.20512*.

Zhiheng Xi, Jixuan Huang, Xin Guo, Boyang Hong, Dingwen Yang, Xiaoran Fan, Shuo Li, Zehui Chen, Junjie Ye, Siyu Yuan, and 1 others. 2025. Critique-rl: Training language models for critiquing through two-stage reinforcement learning. *arXiv preprint arXiv:2510.24320*.

Zhiheng Xi, Dingwen Yang, Jixuan Huang, Jiafu Tang, Guanyu Li, Yiwen Ding, Wei He, Boyang Hong,

Shihan Do, Wenyu Zhan, and 1 others. 2024. Enhancing llm reasoning via critique models with test-time and training-time supervision. *arXiv preprint arXiv:2411.16579*.

Zhihui Xie, Liyu Chen, Weichao Mao, Jingjing Xu, Lingpeng Kong, and 1 others. 2025. Teaching language models to critique via reinforcement learning. *arXiv preprint arXiv:2502.03492*.

Jianhao Yan, Yafu Li, Zican Hu, Zhi Wang, Ganqu Cui, Xiaoye Qu, Yu Cheng, and Yue Zhang. 2025. Learning to reason under off-policy guidance. *arXiv preprint arXiv:2504.14945*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025a. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Wenkai Yang, Jingwen Chen, Yankai Lin, and Ji-Rong Wen. 2025b. Deepcritic: Deliberate critique with large language models. *arXiv preprint arXiv:2505.00662*.

Wenkai Yang, Weijie Liu, Ruobing Xie, Yiju Guo, Lulu Wu, Saiyong Yang, and Yankai Lin. 2025c. Laser: Reinforcement learning with last-token self-rewarding. *arXiv preprint arXiv:2510.14943*.

Wenkai Yang, Shuming Ma, Yankai Lin, and Furu Wei. 2025d. Towards thinking-optimal scaling of test-time compute for LLM reasoning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Shiji Song, and Gao Huang. 2025. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *arXiv preprint arXiv:2504.13837*.

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

Kaiyi Zhang, Ang Lv, Jinpeng Li, Yongbo Wang, Feng Wang, Haoyuan Hu, and Rui Yan. 2025a. Stephint: Multi-level stepwise hints enhance reinforcement learning to reason. *arXiv preprint arXiv:2507.02841*.

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

Yifan Zhang and Team Math-AI. 2024. American invitational mathematics examination (aime) 2024.

Yifan Zhang and Team Math-AI. 2025. American invitational mathematics examination (aime) 2025.

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

Yuxin Zuo, Kaiyan Zhang, Li Sheng, Shang Qu, Ganqu Cui, Xuekai Zhu, Haozhan Li, Yuchen Zhang, Xinwei Long, Ermo Hua, and 1 others. 2025. Ttrl: Test-time reinforcement learning. *arXiv preprint arXiv:2504.16084*.

A Prompt Templates

A.1 Prompt Templates for Main Experiments

Critique Prompt Template

You are given a problem and an AI-generated solution.

Problem:
{question}

AI Solution:
{solution}

Your tasks:
1) Carefully read Problem and the entire AI Solution from start to finish.
2) Identify the single earliest step or statement whose error *affects the final answer*.
- Ignore any corrected or irrelevant intermediate errors.
- If the final answer is missing, this must be considered an error.
- If multiple issues propagate, choose the earliest one.
- If no errors propagate to the final answer, set `ERROR_FOUND: false` and proceed to task 5.

3) Quote the *minimal* snippet that contains this issue.
4) Briefly explain why this issue affects the final answer.
5) Produce a high-level hint that helps a student avoid making the *same kind of error* (if any error was found) or *ensure correctness* (if the solution was flawless).
- Do NOT provide the next step or the correct answer.
- The hint must be strategic and grounded in the problem's context, but avoid referencing specific numbers, symbols, or lines from the solution.

Rules:
- Do not rewrite or solve the problem.
- The hint must be self-contained and avoid specific computations or targeted fixes.
- Even when `ERROR_FOUND: false`, you should still provide a general hint relevant to the overall approach.
- You must output all four fields below in order.

Output format:
ERROR_FOUND: true/false
ERROR_QUOTE: <the minimal snippet or "">
WHY_IT_MATTERS: <1–2 sentence explanation or "">
HIGH_LEVEL_HINT: <general strategy hint without specific steps or answers>

Critique-Guided Generation Prompt Template

You are given a Problem and a Hint.

Task:
1. Generate a new solution to the Problem.
2. Carefully incorporate the guidance from the Hint, without making any explicit reference to the Hint in your output.
3. Ensure the reasoning is consistent, clear, and logically sound.
4. The final output should be a complete step-by-step solution to the Problem.

Input:
Problem:
{problem}

Hint:
{hint}

Output:
<your solution here, put your final answer within `\\boxed{{}}`>

A.2 Prompt Templates for Pilot Study

Prompt Template for Generating High Level Hint

You are given a problem and an AI-generated solution.

Problem:
{problem}
Please reason step by step, and put your final answer within `\\boxed{{}}`.

AI Solution:
{initial response}

Your tasks:
1) Carefully read Problem and the entire AI Solution from start to finish.
2) Identify the single earliest step or statement whose mistake survives to the final answer (i.e., it is NOT fully corrected later).
- If an earlier mistake is later corrected and no longer affects the final answer, ignore it.
- If multiple issues survive, choose the earliest one.
3) Quote the minimal snippet that contains this issue.
4) Briefly explain why this issue affects the final answer.
5) Produce a high-level, general hint that helps a student avoid making the same kind of mistake.
- Do NOT provide the next step or the correct answer.
- Keep the hint general and transferable, not tied to specific numbers, symbols, or lines.

Rules:
- Do not rewrite or solve the problem.
- The hint should be strategic, 1–3 sentences, and avoid specific computations or targeted fixes.

Output format:
ERROR_QUOTE: <the minimal snippet or " " >
WHY_IT_MATTERS: <1–2 sentence explanation>
HIGH_LEVEL_HINT: <general strategy hint without specific steps or answers>

Prompt Template for Hint-guided Re-exploration

You are given a Problem and a Hint.

Task:
1. Generate a new solution to the Problem.
2. Carefully incorporate the guidance from the Hint.
3. Ensure the reasoning is consistent, clear, and logically sound.
4. The final output should be a complete step-by-step solution to the Problem.

Input:
Problem:
{problem}

Hint:
{hint}

Output:
<your improved solution here, put your final answer within `\\boxed{{}}`>

Prompt Template for Generating Specific Critique

You are a mathematics expert. A student is trying to solve a question, and the student's solution contains mistakes.

Task:
Briefly and step-by-step determine whether the student's solution is correct.
If you find an error, ONLY point out where the error is and why it is wrong.
Do NOT continue solving the problem, do NOT provide the correct method, and do NOT give the correct final answer.

Question: **{problem}**

Student's Solution: **{initial response}**

Critique:

Prompt Template for Specific Critique-guided Re-exploration

You are given a question, a previous wrong solution, and a critique that identifies where the previous solution went wrong.
Your task is to write a fully revised, correct solution.

Inputs:
Question:
{problem}

Previous Solution:
{initial response}

Critique:
{critique}

Please re-answer by:
- Use the critique to fix all identified mistakes. Ensure the revised solution addresses all issues raised in the critique.
- Provide a complete, clear, step-by-step solution from scratch.
- Do NOT quote or restate the previous solution. Do NOT reproduce the incorrect steps.
- Placing your final answer within `\\boxed{{}}`.

B Usage of AI Assistants

We used AI assistants (specifically Gemini 3) solely for the purpose of refining the clarity, grammar, and flow of the manuscript's writing. All scientific concepts, experimental designs, data analyses, and conclusions presented in this work are original and were conducted by the authors.

C Pilot Study

To investigate how different re-exploration strategies influence model performance, we conducted a pilot experiment comparing four critique-guided configurations against a re-sampling baseline. The four guided setups—Self-Specific, Self-Hint, Teacher-Specific, and Teacher-Hint—are constructed by crossing two models (Self vs. Teacher) with two strategies (Task-specific vs. High-level hints). For each question, we first sampled four candidate responses from the policy model and evaluated their correctness. Focusing on the zero-pass subset, we apply our four guided configurations. Each configuration generates two independent critiques, each leading to four refined responses, totaling eight refinements. These are compared against a Re-Sampling baseline of eight direct samples without feedback.

The evaluation metrics include SameRatio and Accuracy. SameRatio measures the proportion of the eight newly generated responses that exactly replicate the originally selected incorrect answer. A higher SameRatio indicates a stronger tendency to persist in the same error during re-exploration. Accuracy is defined as the overall correctness rate aggregated across both the initial responses and the newly generated responses.

We conducted our pilot experiments using the Qwen2.5-7B-Instruct and Qwen3-4B models across three challenging mathematical benchmarks: MATH, AIME25, and OlympiadBench. We use Qwen2.5-72B-Instruct as the teacher model. For the Qwen2.5-7B-Instruct model, we set the temperature to 1 and top-p to 0.95. For the Qwen3-4B model, the temperature was set to 0.7 and top-p to 0.8.

As illustrated in Figure 6, across multiple datasets and both evaluated models, the Specific Critique Guide yielded a high SameRatio and lower Accuracy than random sampling, even with prompts explicitly identifying errors. This validates the tendency for models to inadvertently replicate incorrect responses. Our approach effectively mitigates this issue, achieving a lower SameRatio and higher Accuracy, which in turn facilitates more efficient re-exploration.

Hyperparameter	Value
Train Batch Size	128
Mini-Batch Size	128
Learning Rate (LR)	1×10^{-6}
KL Coefficient (β)	0.0
Rollout (K)	8
Temperature	1.0
Top-p	1.0
Max Prompt Length	6144
Max Response Length	4096

Table 4: Detailed training hyperparameters for both CURE and the GRPO baseline.

Hyperparameter	Value
Critique Batch Size	256
Rollout (Initial K)	4
Rollout (Guided M)	4
Rollout (Critique)	1
Format Penalty (λ_1, Missing Keys)	0.5
Format Penalty (λ_2, Length)	0.25

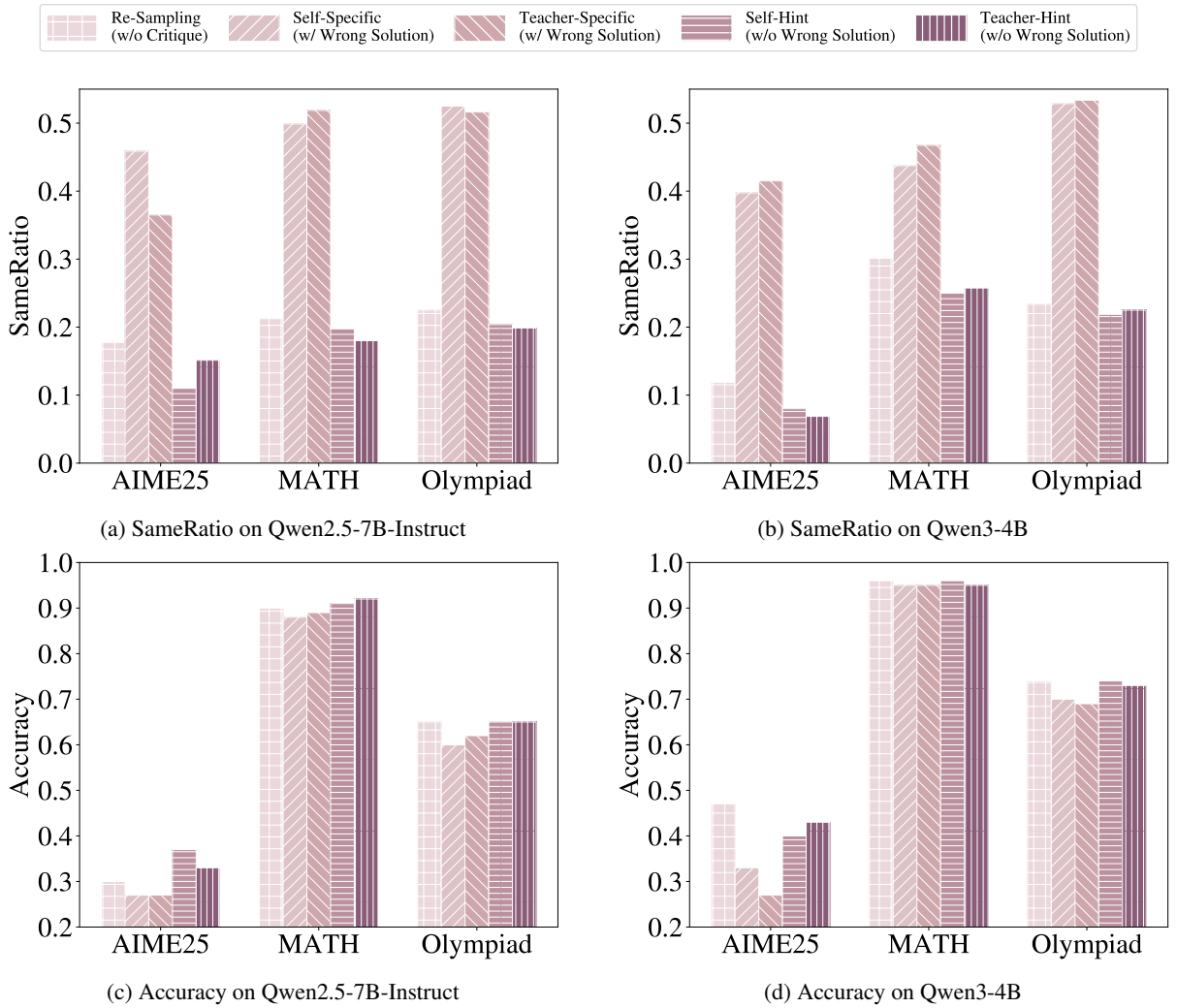
Table 5: CURE-specific hyperparameters.

D Training Details

D.1 Dataset Construction

Mathematical Reasoning To construct a training set conducive to complex reasoning, we preprocess the DeepMath (He et al., 2025) dataset by rigorously filtering out binary classification problems (e.g., True/False questions). These instances are excluded because their simplistic answer space makes them highly susceptible to random guessing, which can yield spurious rewards that poorly reflect actual reasoning ability—thereby introducing noise into the RL signal. From the remaining pool, we randomly downsample 76,800 instances. Given a global training batch size of 128, this dataset size corresponds to exactly 600 training steps.

Code Generation For the coding domain, we follow the setup in Wang et al. (2025b), utilizing their curated subset of CodeContests which contains approximately 4.5k training instances.



(a) SameRatio on Qwen2.5-7B-Instruct

(b) SameRatio on Qwen3-4B

(c) Accuracy on Qwen2.5-7B-Instruct

(d) Accuracy on Qwen3-4B

Figure 6: Comparison of SameRatio and Accuracy across different re-exploration strategies on Qwen2.5-7B-Instruct and Qwen3-4B models.

D.2 Optimization and Hyperparameters

Both CURE and the GRPO baseline are implemented using the `verl` framework and optimized via AdamW. For CURE, parameter updates are performed once per step using a unified batch that aggregates data from initial solutions, critiques, and guided solutions.

For mathematical tasks, we train for 600 steps with a global batch size of 128. For coding tasks, we train for a total of 300 steps with a global batch size of 32. All other optimization hyperparameters remain consistent with the math experiments. For the **Qwen3-4B-Base** model, we employ a brief warmup phase: we first train using standard GRPO for 50 steps to stabilize instruction following, and then use this checkpoint to initialize CURE training.

Table 4 lists the common training hyperparameters shared by both methods. Additionally, Table 5 details the specific hyperparameters introduced by our CURE framework. All experiments were conducted on a server equipped with 4 NVIDIA A800 GPUs.

D.3 Training Cost Analysis

Compared with standard RLVR, CURE introduces additional training overhead due to its multi-stage generation process, which includes initial solution generation, critique generation, and critique-guided re-generation within each optimization step. To quantify this overhead, we report the wall-clock training cost measured on mathematical tasks using Qwen2.5-7B-Instruct on a server with 4 NVIDIA A800 GPUs.

Table 7 compares the per-step cost of the GRPO baseline and CURE. Rollout Time denotes the to-

Method	Mathematical Reasoning						Code Generation			
	AIME24	AIME25	AMC	MATH	Olym.	Avg.	MBPP	LCB	CF	Avg.
Qwen2.5-7B-Instruct										
Base Model	44.5	36.8	68.7	64.5	64.5	55.8	18.0	62.2	60.5	46.9
GRPO	53.3	38.3	73.0	76.3	73.6	62.9	16.3	64.3	47.0	42.5
CURE (Ours)	**72.4**	**80.7**	**75.5**	**77.9**	**74.9**	**76.3**	**49.7**	**68.3**	**60.6**	**59.4**
Qwen3-4B-Base										
Base Model	22.9	23.9	25.5	25.9	24.8	24.6	20.0	18.6	17.8	18.8
GRPO	42.2	50.4	49.7	56.9	46.8	49.2	**52.0**	54.3	35.9	47.4
CURE (Ours)	**67.9**	**77.3**	**71.4**	**71.5**	**71.3**	**71.9**	51.5	**68.9**	**55.6**	**58.7**

Table 6: Self-verification performance (F1 score, %) on mathematical reasoning and code generation benchmarks.

Metric	GRPO	CURE (Ours)
Rollout Time	∼70.2s	∼117.1s
Total Step Time	∼191.0s	∼267.6s

Table 7: Training cost comparison per optimization step on mathematical tasks using Qwen2.5-7B-Instruct with 4 NVIDIA A800 GPUs.

tal time spent on response generation within one training step. For CURE, this includes all three generation phases. Total Step Time denotes the end-to-end wall-clock time of one complete training iteration, including both rollout and optimization.

As shown in Table 7, CURE increases rollout time relative to GRPO due to its richer training pipeline, while the increase in overall step time is more moderate. We note that this overhead can be alleviated in practice through pipelined execution across batches, where solution generation, critique generation, and guided re-generation are overlapped.

E Evaluation Details

During inference, we set the sampling temperature to 0.6 and the maximum generation length to 8192 tokens across all tasks. For the code generation benchmark LiveCodeBench, we utilize Version 2, which consists of 511 problems.

To evaluate the test-time self-improvement potential, we implement an iterative self-improvement loop with a maximum depth of $T = 8$ rounds. The process proceeds as follows for each query:

1. **Initialization:** The model generates an initial solution $\boldsymbol{y}_0$ using the standard sampling parameters.

2. **Iterative Re-generation:** In each round $t \in [1, T]$:

- The model generates a critique $\boldsymbol{c}_t$ for the current solution $\boldsymbol{y}_{t-1}$.
- We parse the verification judgment from $\boldsymbol{c}_t$. If the judgment is `"Error_Found: False"` (i.e., predicted correct), the loop effectively pauses, and the current solution $\boldsymbol{y}_{t-1}$ is carried forward as the result for all subsequent rounds up to T.
- If the judgment is `"Error_Found: True"` (i.e., predicted incorrect), the model utilizes the generated hint $\boldsymbol{h}_t$ to produce a new guided solution $\boldsymbol{y}_t$.

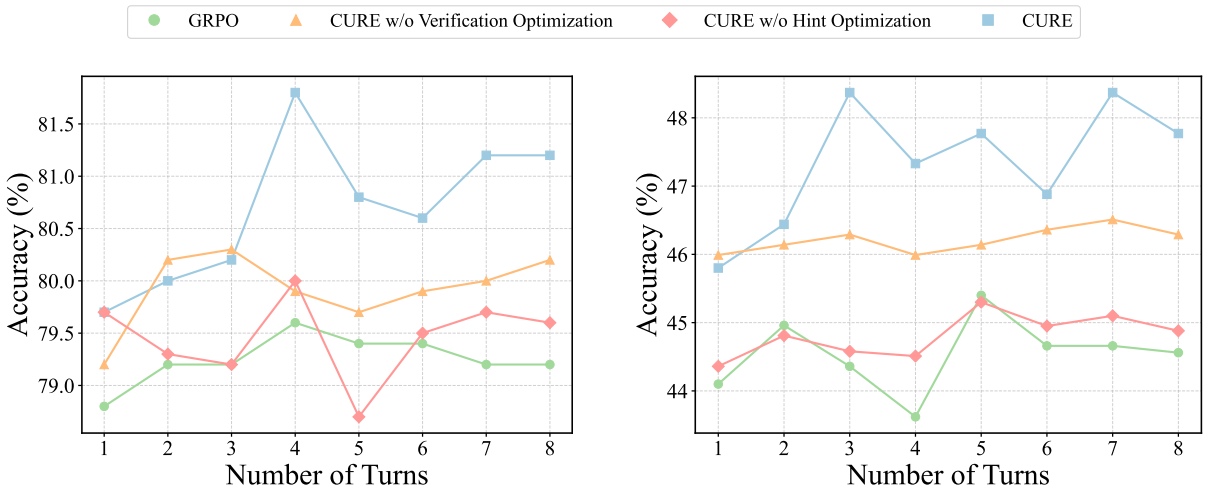
3. **Performance Aggregation:** For smaller benchmarks (AMC and AIME), we perform this scaling process in parallel over 32 independent trajectories and report the average accuracy (Avg@32) at each round t. For larger benchmarks (MATH, Olympiad, and all code tasks), we report Avg@2.

F Additional Results

F.1 Detailed Self-Verification Results

We evaluate the self-verification capability of the models using the F1 score. Specifically, the F1 score is computed as the harmonic mean of verification accuracy on self-generated correct and incorrect solutions. A higher F1 score indicates a stronger ability to distinguish correct answers from incorrect ones, which is a prerequisite for effective inference-time self-improvement.

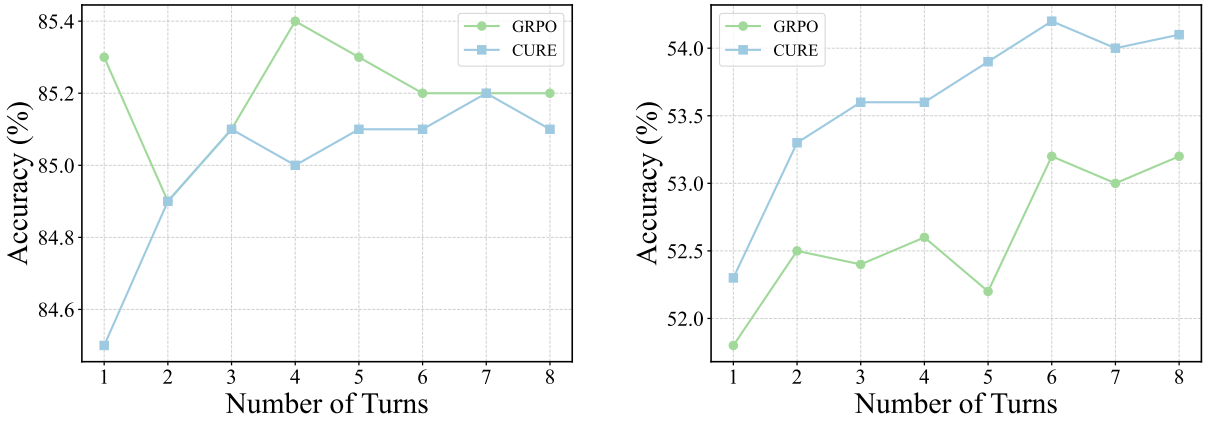
Table 6 reports self-verification performance across all benchmarks. CURE consistently outperforms the GRPO baseline on both mathematical reasoning and code generation tasks. These results demonstrate that our unified training objective effectively grounds critique generation in factual correctness.



(a) Unconditional scaling on **MATH500**.

(b) Unconditional scaling on **OlympiadBench**.

Figure 7: **Unconditional self-improvement scaling results** on mathematical benchmarks using Qwen2.5-7B-Instruct. The plots track accuracy changes over $T = 8$ iterative rounds without verification filtering.



(a) Results on **MATH500**.

(b) Results on **OlympiadBench**.

Figure 8: Inference-time scaling results on Qwen3-4B-Base.

F.2 Analysis of Unconditional Scaling Strategy

In the main text, we presented results using the *conditional scaling* strategy (Strategy A), where the model only re-generates a solution if it predicts an error. Here, we analyze an alternative **Unconditional Re-generation** strategy, where the model iteratively generates a new solution based on the critique at every round, regardless of the verification judgment.

Figure 7 illustrates the scaling trajectories of this unconditional strategy on MATH500 and OlympiadBench using Qwen2.5-7B-Instruct. We observe two key phenomena:

1. Initial Improvement: Similar to the conditional re-generation approach, the unconditional strategy achieves accuracy gains in the first 3-4 rounds. This confirms that the generated hints are intrinsically constructive and can guide the model towards better solutions even without explicit filtering.
2. Instability in Later Stages: Unlike the conditional strategy which maintains a stable peak, the unconditional strategy exhibits performance oscillation after the initial rise. This instability arises because the model lacks a stopping mechanism, leading it to more frequently overwrite correct solutions with incorrect ones in later turns than the conditional strategy.

Comparing these strategies highlights the importance of both capabilities in CURE: while the hint provides the necessary gradient for upward improvement (Exploration), the verification capability acts as a crucial gatekeeper (Exploitation), locking

Method	Acc. (%)	Avg. Tokens	Latency (s)
GRPO + BoN (N=8)	48.2	∼10678	∼**0.66**
CURE + Iter. (T=8)	**48.7**	∼**4211**	∼0.95

Table 8: Efficiency and latency trade-off on OlympiadBench under the same maximum generation budget. Avg. Tokens denotes the total number of generated tokens per query. For BoN, we conservatively exclude the additional latency of the external reward model used for reranking.

in correct solutions to convert volatile gains into a stable trajectory.

F.3 Additional Scaling Results

Additionally, we provide the scaling trajectories for the Qwen3-4B-Base model in Figure 8. Consistent with the Qwen2.5 results, CURE demonstrates robust self-improvement on both MATH500 and OlympiadBench.

F.4 Efficiency and Latency on OlympiadBench

To complement the main-text analysis on MATH500, we further compare GRPO + Best-of-N (N=8) and CURE + iterative self-improvement (T=8) on OlympiadBench. Table 8 reports the accuracy, average generated tokens per query, and latency per query. Similar to the MATH500 results, CURE achieves slightly better accuracy while using substantially fewer generated tokens. However, because OlympiadBench is more challenging and often requires more refinement turns before early stopping, the sequential iterative process leads to higher latency in this setting.

F.5 Results on an Additional Backbone Model

To further evaluate the robustness of CURE across model families, we additionally conduct experiments on OctoThinker-3B-Short-Base (Wang et al., 2025c), a LLaMA-based model that undergoes mid-training. The results are reported in Table 9.

Due to the relatively limited capacity of this 3B model, it produces too few correct solutions on the challenging AIME24 and AIME25 benchmarks for stable evaluation. We therefore focus on three representative benchmarks: MATH500, OlympiadBench, and AMC23.

As shown in Table 9, CURE maintains competitive single-turn performance over the GRPO baseline and achieves stronger inference-time self-improvement under iterative scaling. In particular, with test-time self-improvement up to $T = 8$ rounds, CURE improves the average accuracy from 25.6% to 26.5%, while the GRPO baseline exhibits almost no gain (25.3% to 25.2%). These results further support that the benefits of CURE generalize beyond the Qwen family to other backbone architectures.

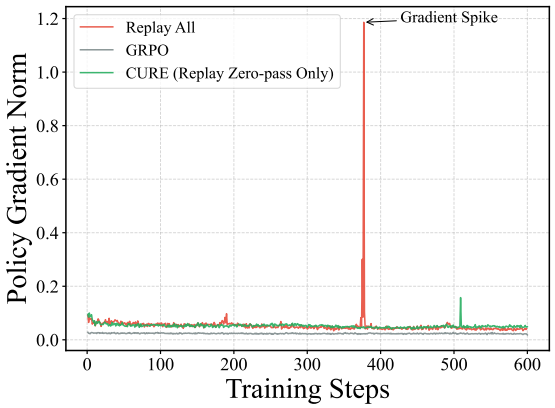


Figure 9: Gradient norm dynamics.

F.6 Impact of Experience Replay Strategy

We investigated two strategies for experience replay: (1) Selective Replay (CURE): Only replaying successful guided solutions from "zero-pass" groups. (i.e., where the model initially failed completely). (2) Full Replay: Replaying all successful guided solutions, including those from groups that already contained correct initial solutions.

As shown in Table 10, while Full Replay yields a marginal scaling improvement (41.8%), it significantly degrades the base single-turn reasoning capability (40.1%) below the GRPO baseline (41.0%). In contrast, Selective Replay achieves optimal performance on both metrics.

We attribute the degradation in Full Replay to excessive distribution shift. Since guided solutions are conditioned on hints (Query + Hint), they deviate from the standard inference distribution. Indiscriminately treating them as standard training samples introduces off-policy noise, destabilizing the optimization landscape. As visualized in Figure 9, the Full Replay strategy (Pink Line) exhibits a sudden spike in the actor's gradient norm, indicating training instability. By restricting replay to "zero-pass" groups, CURE injects high-value learning signals only where they are strictly necessary, preserving stability while breaking exploration stagnation.

Method	MATH500	OlympiadBench	AMC23	Avg.
Base Model	3.5	1.2	1.1	1.9
GRPO (Baseline)	41.8	13.2	20.9	25.3
+ Self-Improve (T=8)	42.2	12.3	**21.1**	25.2
CURE (Ours)	42.0	14.3	20.4	25.6
+ Self-Improve (T=8)	**43.6**	**14.8**	21.0	**26.5**

Table 9: Results on OctoThinker-3B-Short-Base.

Method	**Single-turn Avg.**	**Scaling Avg. (T=8)**
GRPO Baseline	41.0	41.4
Full Replay	40.1	41.8
Selective Replay (CURE)	**42.0**	**44.9**

Table 10: Ablation study on Experience Replay Strategy (Average accuracy across 5 math benchmarks).

G Licenses

Datasets and Benchmarks. The training dataset is derived from DeepMath (MIT License) and CodeForces (CC-BY-4.0). We evaluate on: MATH 500 (MIT License), AIME 2024/2025 (CC0: Public Domain), AMC 2023 (Apache License 2.0), Olympiad Bench (MIT License), CodeForces (CC-BY-4.0), MBPP (CC-BY-4.0), and LiveCodeBench (CC License).

Framework. RL training is based on verl v0.5 (Apache-2.0 license). Evaluation is performed using vllm framework (Kwon et al., 2023) v0.8.5 (Apache-2.0 License).

H Qualitative Comparison of Hint Quality

Figure 10 presents a qualitative comparison between CURE and the GRPO baseline on a number theory problem. The initial solution correctly identifies the smallest solution $(1, 2, 3)$ but fails to consider permutations. The GRPO baseline model, lacking explicit critique training, generates a hint that is technically sounding but semantically irrelevant to the actual error. Consequently, the guided re-generation fails to break out of the initial incorrect path, repeating the same incomplete answer. In contrast, CURE generates a strategic hint that directly challenges the flaw in the reasoning logic. Crucially, this hint does not leak the answer but instead prompts the model to reconsider its assumptions. It successfully triggers a new line of reasoning ("Symmetry and Permutations"), enabling the model to discover the complete set of solutions. This case exemplifies how CURE's critique capability acts as an effective scaffold for escaping local optima.

Case Study

Problem:
Find all triples (a, b, c) of positive integers such that $a^3 + b^3 + c^3 = (abc)^2$.

Initial (Incorrect) Solution:
To find all triples (a, b, c) of positive integers such that...
[...Intermediate steps omitted...]
Given the complexity and the rapid growth of the terms, we can infer that the only solution found is the smallest one.
Thus, the only solution is: $\boxed{(1, 2, 3)}$ **(Incorrect)** – Overlooked permutations

GRPO

Generated Hint:
[...other fields omitted...]
Always verify the behavior of polynomials for positive integer solutions by testing small values and considering the polynomial's derivative to understand its trend.

New Solution (Guided by Baseline):
[...Intermediate steps omitted...]
After testing small values and analyzing the behavior of the polynomial, we find that the only solution is $\boxed{(1, 2, 3)}$ **(Still Incorrect)**

Ours

Generated Hint:
[...other fields omitted...]
Always be cautious with the assumption that the smallest or first solution found is the only one, especially when dealing with polynomial equations.

New Solution (Guided by Ours):
[...Intermediate steps omitted...]
Thus, $c = 3$ is a solution, confirming $(1, 2, 3)$ is a solution since $1 + 8 + 27 = 36 = 6^2$.

Symmetry and Permutations: *← Logic triggered by Hint*
Since the equation is symmetric in a, b, c, we generally test permutations of the found solution $(1, 2, 3)$.
All the following are solutions:

$$(1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1)$$

[...Proof for no larger solutions omitted...]

Conclusion:
The solutions are all permutations of $(1, 2, 3)$.
Final Answer:

$$\boxed{(1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1)}$$

(Correct)

Figure 10: Qualitative comparison of generated hints.