Save the Good Prefix: Precise Error Penalization via Process-Supervised RL to Enhance LLM Reasoning

Haolin Liu[1,2] Dian Yu[1] Sidi Lu[1] Yujun Zhou[1,3] Rui Liu[1,4]
Zhenwen Liang[1] Haitao Mi[1] Chen-Yu Wei[2] Dong Yu[1]
[1]Tencent AI Lab, Seattle [2]University of Virginia [3] University of Notre Dame
[4] University of Maryland

Abstract

Reinforcement learning (RL) has emerged as a powerful framework for improving the reasoning capabilities of large language models (LLMs). However, most existing RL approaches rely on sparse outcome rewards, which fail to credit correct intermediate steps in partially successful solutions. Process reward models (PRMs) offer fine-grained step-level supervision, but their scores are often noisy and difficult to evaluate. As a result, recent PRM benchmarks focus on a more objective capability: detecting the first incorrect step in a reasoning path. However, this evaluation target is misaligned with how PRMs are typically used in RL, where their step-wise scores are treated as raw rewards to maximize. To bridge this gap, we propose Verifiable Prefix Policy Optimization (VPPO), which uses PRMs only to localize the first error during RL. Given an incorrect rollout, VPPO partitions the trajectory into a verified correct prefix and an erroneous suffix based on the first error, rewarding the former while applying targeted penalties only after the detected mistake. This design yields stable, interpretable learning signals and improves credit assignment. Across multiple reasoning benchmarks, VPPO consistently outperforms sparse-reward RL and prior PRM-guided baselines on both Pass@1 and Pass@K.

1 Introduction

Eliciting complex, multi-step reasoning from large language models (LLMs) remains a fundamental challenge. Current reinforcement learning (RL) methods such as GRPO (Shao et al., 2024) mostly rely on sparse, binary outcome rewards (i.e., success or failure at the final answer), which creates a severe credit assignment problem by discarding all intermediate progress when the final answer is incorrect. Process Reward Models (PRMs) were introduced to address this gap by evaluating each intermediate step (Lightman et al., 2023). However, when used within RL, they introduce a new bottleneck of reward signal quality. Most existing PRM-based methods (Wang et al., 2024; Setlur et al., 2024; Gao et al., 2024; Cui et al., 2025a; Zou et al., 2025) treat the PRM score as a step-wise reward and optimize the cumulative return over the entire trajectory. In practice, process rewards are often highly biased (Zheng et al., 2025a; Zhang et al., 2025b), and it is intrinsically difficult to interpret whether a given score is good or bad because ground-truth step-wise scores are rarely well defined. By contrast, current benchmarks for PRMs (Zheng et al., 2024; Song et al., 2025a) typically focus on a more objective task: *finding the first incorrect step*. This metric admits clear ground truth and enables training PRMs that can accurately localize the first error. These observations naturally raise the question: *Can we rely solely on information about the first incorrect step of a response to enhance LLM reasoning in RL?*

We begin by showing that the portion of a failed trajectory before its first error is often valuable. We define the *good prefix* as the sequence of correct reasoning steps up to (but not including) the first error. In Figure 1, we analyze 3,000 randomly sampled incorrect rollouts collected during training. We find that 88% of incorrect responses contain at least one correct step, and that, on average, the *good prefix* constitutes 34% of the full reasoning trajectory. To assess the usefulness of these *good prefixes*, we conduct a preliminary study on AIME25. For each question, we first sample 32 complete responses, retain only the incorrect ones, and extract their good prefixes. We then sample 32 continuations conditioned on each extracted *good prefix*. We find 8 out of 30 questions for which the model initially fails across all 32 attempts, yet at least one extracted *good prefix* yields a correct solution among its 32 sampled continuations. This result suggests that, for difficult problems, failure often arises not from a lack of underlying knowledge, but from the derailment of an otherwise valid reasoning trajectory

Findings of the Association for Computational Linguistics: ACL 2026, pages 35450–35477

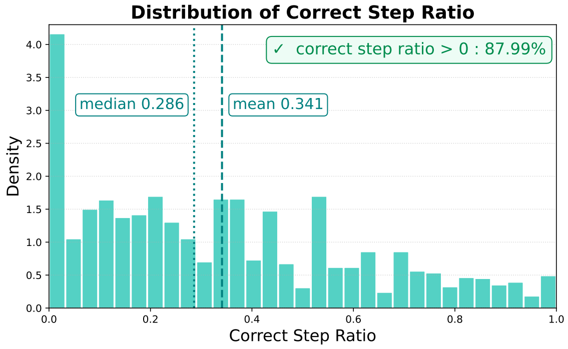


Figure 1: **Distribution of Correct Steps before the First Error in Incorrect Responses.** Among incorrect trajectories, 88% contain at least one correct step before the first error. We define the correct-step ratio as the number of steps before the first incorrect step divided by the total number of steps; the average correct-step ratio is 34%.

(see Appendix B for details). Overall, these findings highlight that failed trajectories often contain substantial correct structure, and that preserving this structure can meaningfully improve success.

Motivated by these observations, we propose to repurpose PRMs in RL solely as first-error detectors and introduce **V**erifiable **P**refix **P**olicy **O**ptimization (**VPPO**), a new RL framework that utilizes a PRM to identify the first incorrect step in an incorrect response, thereby partitioning the trajectory into a good prefix and an incorrect suffix. VPPO then assigns reward to the final token of the good prefix before the first incorrect step. Since first-error detection is well supported by current benchmarks such as ProcessBench (Zheng et al., 2024), this simple shaping scheme provides a reliable dense reward signal: it preserves and reinforces good prefixes while specifically penalizing incorrect continuations. We show that VPPO mitigates key exploration–exploitation failures of sparse, outcome-only rewards and prevents gradient vanishing in the sparse-reward regime when all sampled rollouts are incorrect. Empirically, VPPO improves both precision (Pass@1) and coverage (Pass@K) on several reasoning benchmarks, consistently outperforming sparse-reward methods and prior PRM-guided approaches.

Our work relates to multiple existing research lines. By rewarding high-quality intermediate steps that the model already produces, our approach is closely connected to subgoal rewards in goal-oriented reinforcement learning (Ng et al., 1999; Huang et al., 2017; Paul et al., 2019; Jurgenson et al., 2020; Okudo and Yamada, 2021; Ibrahim et al., 2024), where partial progress toward a goal is explicitly incentivized to ease credit assignment. In LLM reasoning, the good prefix can be viewed as a sequence of achieved subgoals along a chain of thought, and rewarding it provides a similarly shaped learning signal. Our method is also related to hint-augmented reasoning (Liu et al., 2025b; Zhang et al., 2025a; Huang et al., 2025b), which appends expert-generated initial steps after a question to guide generation of small models. In our approach, the good prefix plays the role of a self-generated hint, and preserving it with additional reward can further speed learning. Additional related work is discussed in Appendix A.

2 Preliminary

We view each model response at both token and step level. A response o is a token sequence $o = (o_1, o_2, \ldots, o_{|o|})$ with length $|o|$. For any token sequence o, we use $\mathrm{LT}(o)$ to denote the *last token* of sequence o. We also segment o into M_o contiguous steps $o = (s_1, s_2, \ldots, s_{M_o})$ where each step s_m is itself a subsequence of tokens $s_m = (o_{t_m}, o_{t_m+1}, \ldots, o_{t_{m+1}-1})$, $1 = t_1 < t_2 < \cdots < t_{M_o+1} = |o| + 1$. Thus every step is a block of consecutive tokens, and the union of all steps recovers the full token sequence.

2.1 Group Relative Policy Optimization (GRPO)

For a given question q, a policy LLM $\pi_{\theta_{\text{old}}}$ generates a group of G complete responses $\{o^1(q), \ldots, o^G(q)\}$. We also simplify $o^i(q)$ to o^i for $i \in [G]$ when the given question is clear or unimportant. For each response $o^i = (o_1^i, \cdots, o_{|o^i|}^i)$ where every entry is a token, we define $o_{<t}^i = (o_1^i, \cdots, o_{t-1}^i)$ and the reward for every token $t \in [|o^i|]$ is defined as

$$r(q, o_{<t}^i, o_t^i) = \begin{cases} 1 & o^i \text{ is correct and } o_t^i = \mathrm{LT}(o) \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

In other words, only the last token of a correct response receive 1 reward, all other tokens receive 0 reward. To calculate the advantage $A_{i,t}(q)$ of token t in response o^i for question q, GRPO leverage standard Monte-Carlo estimation such that

$$A_{i,t}(q) = \begin{cases} 1 & o^i \text{ is correct} \\ 0 & \text{otherwise} \end{cases}$$

To stabilize training, GRPO utilize group-level normalization by first calculating the average token-advantage for every response. For every $i \in [G]$,

$$A_i(q) = \frac{1}{|o^i|}\sum_{t=1}^{|o^i|} A_{i,t}(q) = \begin{cases} 1 & o^i \text{ is correct} \\ 0 & \text{otherwise} \end{cases}$$

The normalized advantage is then calculated through Eq. (2).

$$\widehat{A}_{i,t}(q) = \frac{A_{i,t}(q) - \text{mean}(A_1(q), \ldots, A_G(q))}{\text{std}(A_1(q), \ldots, A_G(q))}. \tag{2}$$

For a given batch b and the set of questions $\mathcal{Q}_b$ in b, with the advantage estimates in Eq. (2), we optimize the policy via the clipped surrogate objective in Eq. (3), where $o_{1:G}(q) \sim \pi_{\text{old}}(\cdot \mid q)$, $r_{i,t}(\theta) = \frac{\pi_\theta(o_t^i|q, o_{<t}^i)}{\pi_{\theta_{\text{old}}}(o_t^i|q, o_{<t}^i)}$, and ϵ is a fixed clip hyperprameter.

$$\frac{1}{|\mathcal{Q}_b|}\sum_{q\in\mathcal{Q}_b}\frac{1}{\sum_{i=1}^{G}|o^i(q)|}\sum_{i=1}^{G}\sum_{t=1}^{|o^i(q)|}\min\Big\{r_{i,t}(\theta)\widehat{A}_{i,t}(q), \\ \text{clip}(r_{i,t}(\theta), 1-\epsilon, 1+\epsilon)\,\widehat{A}_{i,t}(q)\Big\}. \tag{3}$$

2.2 Current Methods to Enhance GRPO with Process Supervision

One limitation of GRPO is that it only leverages the final sparse reward to provide update signal, making it hard to distinguish the quality of responses that have the same correctness label. To provide fine-grained signal, previous works tried to utilize an external process reward model to provide the assessment of step-level quality. We next describe two prior methods that, like ours, directly use an external process reward model to guide RL without any online reward-model training, and we adopt them as our baselines.

2.2.1 Algorithm in (Zou et al., 2025).

For a given question q and a reasoning response $o = (s_1, \cdots, s_{M_o})$ where $s_{1:M_o}$ denote M_o steps in the reasoning path o. Zou et al. utilizes a process reward model r_{prm} to score every step and get $r_{\text{prm}}(q, s_{<m}, s_m)$ for every step $m \in [M_o]$. Together with the final correctness reward $r_{\text{corr}}(q, o)$, which is 1 if o is correct otherwise is 0, the final reward of a response o for question q is defined as $r_{\text{mix}}(q, o) = \frac{\lambda}{M_o}\sum_{m=1}^{M_o} r_{\text{prm}}(q, s_{<m}, s_m) + (1 - \lambda) r_{\text{corr}}(q, o)$, where λ is a hyperparameter to control the mixture rate.

This reward r_{mix} combine the average process value and the correctness label, which is also only given to the last token of response o. Thus, for a question q and a group of responses $\{o^1, \cdots, o^G\}$, the Monte-Carlo advantage estimation for token t in response o^i is $A_{i,t}^{\text{mix}}(q) = r_{\text{mix}}(q, o^i)$. Following the normalization of GRPO, the token-average advantage of response o^i is $A_i^{\text{mix}}(q) = \frac{1}{H_i}\sum_{i=1}^{H_i} A_{i,t}^{\text{mix}}(q) = r_{\text{mix}}(q, o^i)$, and the final estimated advantage is

$$\widehat{A}_{i,t}^{\text{mix}}(q) = \frac{A_{i,t}^{\text{mix}}(q) - \text{mean}(A_1^{\text{mix}}(q), \ldots, A_G^{\text{mix}}(q))}{\text{std}(A_1^{\text{mix}}(q), \ldots, A_G^{\text{mix}}(q))}. \tag{4}$$

The estimated advantage in Eq. (4) is then used in loss Eq. (3) for policy update.

2.2.2 Algorithm in (Chen et al., 2025a).

For a given question q and a reasoning response $o = (s_1, \cdots, s_{M_o})$ where $s_{1:M_o}$ denote M_o steps in the reasoning path. Chen et al. utilizes a large external generative models to find the first incorrect step in a incorrect response. Assuming an incorrect response o has M_o steps and the first G_o steps are correct, Chen et al. defines $\text{RTS}(o) = \frac{G_o}{M_o}$ as the fraction of correct steps in an incorrect response. With this value, the final reward is defined as

$$r_{\text{rts}}(q, o) = \begin{cases} 1 & o \text{ is correct} \\ \dfrac{1}{1 + \exp(\beta \text{RTS}(o) + \gamma)} & \text{otherwise} \end{cases}$$

where β and γ are two hyperparameters. With this reward scheme, the incorrect response with more correct steps receive higher reward. Similar to (Zou et al., 2025), this reward r_{rts} is also only given to the last token of response o, thus, for a question q and a group of response $\{o^1, \cdots, o^G\}$, the the Monte-Carlo advantage estimation for token t in response o^i is $A_{i,t}^{\text{rts}}(q) = r_{\text{rts}}(q, o^i)$. Following the normalization of GRPO, the token-average advantage of response o^i is $A_i^{\text{rts}}(q) = \frac{1}{H_i}\sum_{i=1}^{H_i} A_{i,t}^{\text{rts}}(q) = r_{\text{rts}}(q, o^i)$. The final estimated advantage is

$$\widehat{A}_{i,t}^{\text{rts}}(q) = \frac{A_{i,t}^{\text{rts}}(q) - \text{mean}(A_1^{\text{rts}}(q), \ldots, A_G^{\text{rts}}(q))}{\text{std}(A_1^{\text{rts}}(q), \ldots, A_G^{\text{rts}}(q))}. \tag{5}$$

The estimated advantage in Eq. (5) is then used in loss Eq. (3) for policy update.

3 Methods

Zou et al. and Chen et al. incorporate process supervision by collapsing step-level signals into a single response-level scalar applied only at the final token. As a result, all tokens in a response share the same advantage, blurring the learning signal because in an incorrect sample, correct early steps and erroneous later steps receive identical credit. Ideally, the correct prefix should be rewarded more than the incorrect suffix. Our method builds on this insight to overcome this limitation of prior approaches.

Through the process reward model, we can find the first incorrect step, and all steps before it are defined as the *good prefix*. For a question q and its incorrect response o, we use $\mathcal{GP}(q, o)$ to denote the *good prefix* of o. Then, we construct a token sequence $\mathcal{RP}(q, o) \subseteq \text{Pref}\left(\mathcal{GP}(q, o)\right)$ to assign additional reward, where $\text{Pref}(\cdot)$ denotes the set of all token-level prefixes of a sequence. In other word, $\mathcal{RP}(q, o)$ is a valid prefix of $\mathcal{GP}(q, o)$; we refer to it as the *reward prefix*. One natural choice is $\mathcal{RP}(q, o) = \mathcal{GP}(q, o)$ but we will show in Section 4 that other choices can lead to better performance. Assume $o = (o_1, \cdots, o_{|o|})$, we define $o_{<t} = (o_1, \cdots, o_{t-1})$ and the reward for every token $t \in [|o|]$ is defined as

$$r(q, o_{<t}, o_t) = \begin{cases} 1 & o \text{ is correct and } o_t = \text{LT}(o) \\ \alpha & o \text{ is incorrect} \\ & \qquad \text{and } o_t = \text{LT}\left(\mathcal{RP}\left(q, o\right)\right) \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

where $0 \le \alpha \le 1$ is a hyperparameter controlling how many rewards should be given to good prefix of incorrect samples. Compared to the reward scheme in GRPO (Eq. (3)), our reward function defined in Eq. (6) only introduces an additional α reward to the last token of $\mathcal{RP}\left(q, o\right)$, ensuring correct steps are well-preserved. With reward scheme in Eq. (6), for a question q and a group of response $\{o^1, \cdots, o^G\}$, the the Monte-Carlo advantage estimation for token t in response o^i becomes

$$A_{i,t}(q) = \begin{cases} 1 & o^i \text{ is correct} \\ \alpha & o^i \text{ is incorrect and } o^i_t \in \mathcal{RP}(q, o^i) \\ 0 & \text{otherwise} \end{cases}$$

We define the average advantage of response o^i as

$$A_i(q) = \frac{1}{|o^i|} \sum_{i=1}^{|o^i|} A_{i,t}(q) = \begin{cases} 1 & o^i \text{ is correct} \\ \dfrac{\alpha|\mathcal{RP}(q, o^i)|}{|o^i|} & \text{otherwise} \end{cases}$$

Since now the advantage is not binary and the standard deviation could have extreme large value in some cases, we remove the standard deviation in the normalization and only substract the mean of advantage. This leads to Eq. (7). Ablation study on standard deviation is provided in Appendix H.

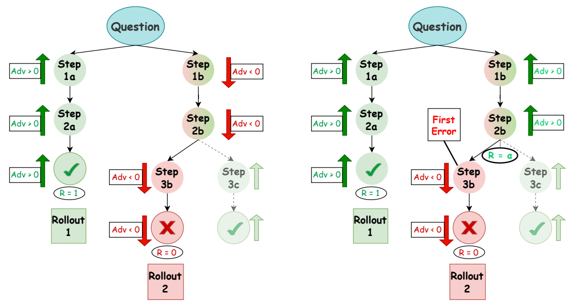
$$\hat{A}_{i,t}(q) = A_{i,t}(q) - \text{mean}(A_1(q), \ldots, A_G(q))\,. \tag{7}$$

We also found that for relatively high-capability model (e.g. instruct-finetuned models), it is better to introduce a RELU operator on the advantage for good prefix to ensure they always have non-negative advantage, leading to

$$\hat{A}_{i,t}(q) = \begin{cases} \max\{A_{i,t}(q) - \text{mean}(A_1(q), \ldots, A_G(q)), 0\} \\ \qquad o^i \text{ is incorrect and } o^i_t \in \mathcal{RP}(q, o^i) \\ A_{i,t}(q) - \text{mean}(A_1(q), \ldots, A_G(q)) \\ \qquad \text{otherwise} \end{cases} \tag{8}$$

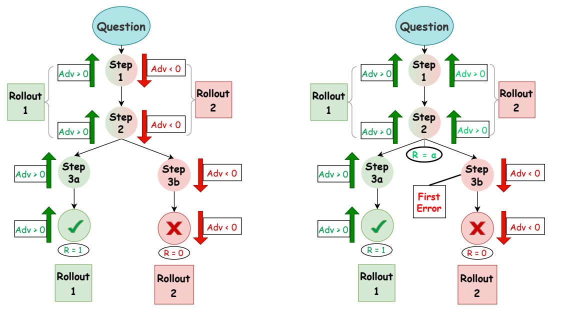
The advantage estimates defined in Eq. (7) and Eq. (8) are subsequently incorporated into the policy-gradient loss in Eq. (3). Our approach is able to leverage opportunities for both exploration and exploitation that standard sparse-reward methods systematically miss. Two illustrative examples are provided in Figure 2 and Figure 3.

In the exploration scenario (Figure 2), two rollouts with different prefixes are observed, denoting as Rollout 1 (with Step $1a$, Step $2a$, and a final correct answer) and Rollout 2 (with Step $1b$, Step $2b$, Step $3b$, and a final incorrect answer). Step $3c$ denotes an *unsampled* continuation after Step $2b$ that would lead to a correct answer. Figure 2a depicts the sparse-reward setting, where the terminal reward R is given only at the end of Rollout 1 and Rollout 2, and the corresponding token-level advantages Adv are shown beside each step (for our method, we use Eq. (8) as an example, where good prefixes do not receive negative advantage). Under sparse rewards, every step in Rollout 1 is encouraged and every step in Rollout 2 is discouraged. However, in Rollout 2, both Step $1b$ and Step $2b$ are in fact good steps that could lead to a correct answer via the (unsampled) Step $3c$. With our reward scheme Eq. (6), since Step $3b$ is the first incorrect step in Rollout 2, Step $1b$ and Step $2b$ are identified as a *good prefix* and receive an additional reward α at the end of Step $2b$ (as shown in Figure 2b). Consequently, Step $1b$ and Step $2b$ are encouraged, while the truly incorrect Step $3b$



(a) Sparse Reward Scheme (b) Our Reward Scheme

Figure 2: **Exploration Example Comparison.** When two rollouts have different prefix, sparse reward only encourages the sampled correct path but our reward scheme enhances the likelihood for more correct paths.



(a) Sparse Reward Scheme (b) Our Reward Scheme

Figure 3: **Exploitation Example Comparison.** When two rollouts have similar prefix but different answers, sparse reward have conflict signal on the shared prefix but our reward scheme can accurately enhance the correct sample.

is discouraged. Discouraging Step $3b$ reduces its log-likelihood and redistributes probability mass to alternative continuations after Step $2b$, implicitly increasing the probability of another correct path [Step $1b$, Step $2b$, Step $3c$]. Compared with sparse reward, our method increases the diversity of the correct output because it leads to the likelihood increase of both Rollout 1 and another correct path [Step $1b$, Step $2b$, Step $3c$]) while sparse rewards only increase the likelihood of Rollout 1. Appendix E.1 shows a case study related to this exploration scenario. In summary, when the sampled rollouts reflect different solution strategies, rewarding the good prefix helps preserve incorrect yet promising attempts. By penalizing only the truly incorrect steps, exploration becomes targeted on the subtree rooted at the good prefix, encouraging the model to recover a correct continuation, even if it corresponds to a solution strategy entirely different from the sampled correct responses. In contrast, sparse outcome rewards discard these partially correct trajectories and instead tend to over-sharpen the policy around the sampled correct solutions.

In the exploitation scenario (Figure 3), the notation is consistent with Figure 2. The two observed rollouts are Rollout 1 (containing Step 1, Step 2, Step $3a$, and a final correct answer) and Rollout 2 (containing Step 1, Step 2, Step $3b$, and a final incorrect answer). These rollouts share the same prefix (Step 1, Step 2). Under sparse rewards (Figure 3a), Steps 1 and 2 are encouraged by Rollout 1 but discouraged by Rollout 2, creating gradient conflict. Under our reward scheme (Eq. (6)), since Step $3b$ is the first incorrect step in Rollout 2, Steps 1 and 2 are identified as *good prefix* and receive reward α at the end of Step 2 (see Figure 3b). Thus, Steps 1 and 2 are consistently encouraged under both rollouts, while only the incorrect Step $3b$ is discouraged. This alignment strengthens the likelihood increase of the correct Rollout 1 compared with sparse-reward methods. See Appendix E.2 for a case study. In summary, when the sampled rollouts share similar early steps that makes meaningful progress, but their later continuations diverge and produce different final answers, rewarding the good prefix fully leverages these shared correct steps while penalizing only the incorrect continuation. In contrast, sparse outcome rewards provide limited learning signal for the strong initial steps due to gradient conflict, making it harder for the policy to reliably benefit from them.

Beyond improved exploration and exploitation, our method can provide informative process reward signals even when all sampled trajectories for a given question are incorrect. This solves the vanishing gradient issue of sparse outcome rewards, where the policy receives no learning signal if every sample is wrong. As a result, sparse-reward methods struggle to learn from hard questions, whereas our process-supervised signal can substantially accelerate progress on difficult problems. Theoretically, we study a simple tree-structured reasoning space in which the correct paths are rare, capturing the challenge of learning on hard questions. In this setting, our reward design yields an **exponential improvement in sample complexity** over sparse rewards for learning a near-optimal policy under standard policy-optimization algorithms. We formalize this result in Theorem 1, with the full proof deferred to Appendix F.

Theorem 1 (Informal)**.** *Consider a H-layer reasoning tree with sparse correct paths (i.e. the question is difficult) where each node denotes a reasoning step. Under standard policy optimization algorithm, let $N^\star_{\textit{sparse}}$ and $N^\star_{\textit{dense}}$ be the minimum sample size to learn a near-optimal policy with sparse reward scheme Eq. (1) and our reward scheme Eq. (6) respectively, we have*

$$\mathbb{E}\left[N^\star_{\textit{sparse}}\right] \geq \tilde{\Omega}\left(\frac{2^H}{H}\right) \quad \textit{and} \quad \mathbb{E}\left[N^\star_{\textit{dense}}\right] \leq \tilde{O}\left(H\right)$$

where $\tilde{\Omega}$ and $\tilde{O}$ omits log *factors.*

4 Experiments

Models and Datasets. We use open-sourced Qwen3-4B-Base, Qwen3-8B-Base and Qwen3-4B (non-thinking) as our base model to finetune. For training data, we utilize axon-rl MATH training data[1]. For the test set, we use AIME-25, AIME-24, AMC-23 (Li et al., 2024a), MATH-500 (Hendrycks et al., 2021), Minerva (Lewkowycz et al., 2022), Olympiadbench(He et al., 2024), Hmmt-feb-2024 and Hmmt-feb-2025 (Balunović et al., 2025).

Experiment Setup. In our discussion on advantage normalization in Section 3, we can either add a RELU operator or not. In our experiment, we found that for base model, it is better to use direct advantage normalization Eq. (7) without RELU, while for high-capability mode (e.g. Qwen3-4B), it is better to use advantage normalization Eq. (8) with RELU. We attribute this to the ability to generate high quality prefix for instruct-tuned models, which should be protected more with RELU to prevent negative gradient. In the following experiments, we set $\alpha = 0.5$ for all models, utilize Eq. (7) advantage normalization for Qwen3-4B-Base and Qwen3-8B-Base, and Eq. (7) advantage normalization for Qwen3-4B. More ablation study is given in Appendix H.

Baseline Methods. We compare our method with group relative policy optimization (GRPO) (Shao et al., 2024), which is the standard method for reinforcement learning for language model reasoning. Besides, we compare with other two methods that utilize process supervision to enhance RL for reasoning as mentioned in Section 2.2. In the following discussion, we will denote the method in (Zou et al., 2025) as **Mixed** and the method in (Chen et al., 2025a) as **RTS**. For Pass@K comparison, we compare with the Pass@K optimization method derived in (Chen et al., 2025b), which is denoted as **PassK** in the follow-up discussion.

First Incorrect Step Identification To enable step-level correctness assessment, we prompt the policy model to explicitly enumerate its reasoning as "Step 1 ...", "Step 2 ...", etc. We do not rely on any initial SFT to enforce this format because the model can learn to follow this step-enumeration pattern as training progresses, producing clearly segmented reasoning traces. We then parse each response into individual steps and score each step using a process reward model (PRM) to identify the first incorrect step. In our implementation, we use Qwen2.5-Math-PRM-7B (Zhang et al., 2025b), which outputs a real-valued score per step. Since our objective is to detect the first incorrect step, we convert step scores into binary correctness labels via an offline threshold selection procedure on ProcessBench (Zheng et al., 2024), a benchmark designed to evaluate PRMs' ability to localize the first error. We find that a threshold of 0.8 performs best: the first step whose PRM score falls below 0.8 is treated as the first incorrect step. More details on threshold selection is in Appendix D. Note that using a small discriminative PRM is lightweight; empirically, this adds only about $1.1\times$ the per-step runtime compared to standard GRPO.

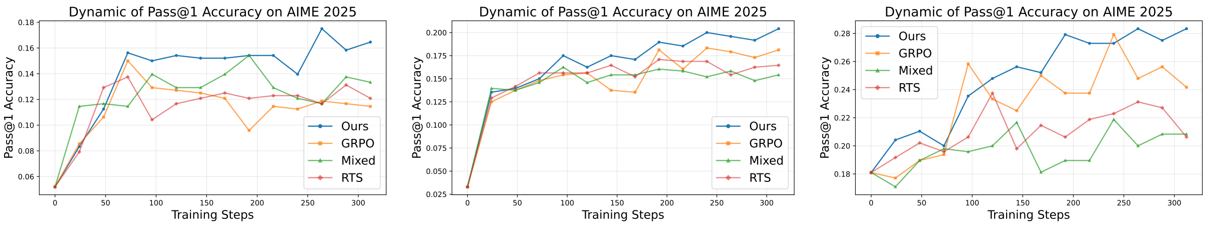
The Choice of $\mathcal{RP}$ in Eq. (6). For a response o with token sequence $(o_1, o_2, \ldots, o_{|o|})$ and step decomposition $(s_1, s_2, \ldots, s_{M_o})$, where each step is a contiguous token subsequence $s_m = (o_{t_m}, o_{t_m+1}, \ldots, o_{t_{m+1}-1})$, the PRM operates at the step level and identifies the index of the first incorrect step $M_{\text{err}} \in \{1, \ldots, M_o\}$ such that $s_1, \ldots, s_{M_{\text{err}}-1}$ are labeled correct and $s_{M_{\text{err}}}$ is the first incorrect step. The standard *good prefix* $\mathcal{GP}(q, o) = (s_1, \ldots, s_{M_{\text{err}}-1}) = (o_1, \ldots, o_{t_{M_{\text{err}}}-1})$ is the entire token span before the first incorrect step. To construct *reward prefix* $\mathcal{RP}(q, o)$, one natural choice is $\mathcal{RP}(q, o) = \mathcal{GP}(q, o)$ and reward the whole *good prefix*. We refer this scheme as **simple-prefix** approach. However, directly reward all good steps causes *reward hacking* via step inflation (see the "simple prefix" curve in Figure 5). Since steps are defined purely by matching the literal marker "Step k", and all tokens before the first incorrect step receive additional reward, the policy can increase its reward by splitting a semantically large step that contains

---

[1] `https://huggingface.co/datasets/axon-rl/MATH-lvl3to5-8k`.

Algorithm	Benchmarks								
	AIME-2025	AIME-2024	AMC-23	MATH-500	Minerva	Olympiadbench	Hmmt-feb-2025	Hmmt-feb-2024	**Average**
					Qwen3-4B-Base				
Base	5.2	7.7	35.9	58.7	21.6	28.7	1.3	2.3	20.2
GRPO	11.9	18.3	59.2	82.3	38.3	47.5	4.0	8.3	33.7
Mixed	12.3	14.9	53.1	80.4	38.5	45.4	3.5	6.0	31.8
RTS	14.0	17.5	54.1	80.6	36.9	46.4	1.3	8.0	32.4
Ours (Shorten Prefix)	**16.6**	**19.4**	**60.0**	**83.0**	**39.1**	**47.7**	**6.0**	**11.0**	**35.4**
					Qwen3-8B-Base				
Base	3.3	6.7	26.4	49.4	23.7	22.4	1.0	1.7	16.8
GRPO	17.2	22.3	63.4	85.2	41.5	51.5	5.4	6.4	36.6
Mixed	15.0	16.7	**64.5**	82.2	**41.7**	48.6	5.6	7.9	35.2
RTS	16.3	15.0	62.8	83.4	39.5	48.6	4.3	**9.5**	35.0
Ours (Shorten Prefix)	**18.3**	**22.7**	63.3	**86.0**	40.4	**52.3**	**12.9**	7.9	**38.0**
					Qwen3-4B (non-think)				
Base	18.1	19.5	66.7	82.5	38.1	51.3	10.0	10.6	37.1
GRPO	26.0	30.4	69.2	**88.5**	40.6	60.0	11.9	16.7	42.9
Mixed	19.6	20.6	69.1	86.2	40.8	55.9	12.3	13.7	39.8
RTS	22.3	31.2	**77.6**	87.8	41.3	59.3	11.8	13.7	43.1
Ours (Shorten Prefix)	**29.2**	**31.8**	74.5	88.0	**41.7**	**60.6**	**12.7**	**19.6**	**44.8**

Table 1: **Average@16 Across Methods and Benchmarks**. The highest number for each column is bold and our method outperforms all baselines.



(a) Qwen3-4B-Base

(b) Qwen3-8B-Base

(c) Qwen3-4B

Figure 4: **Training Dynamics of Test Accuracy on AIME2025.** Our method outperform all baselines during training.

error into many shorter steps. This manipulation moves substantial content that would otherwise belong to the incorrect large step into shorter correct steps that get positive reward, resulting in overly fine-grained and verbose intermediate reasoning that is misaligned with our goal.

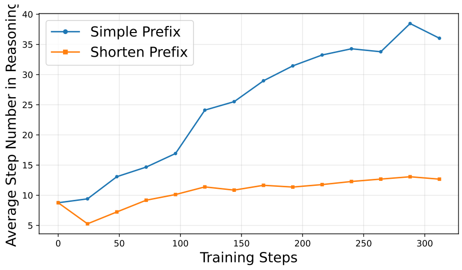


Figure 5: **Training Steps and Reasoning Step Number.** The x-axis shows the training step, and the y-axis reports the average number of reasoning steps in responses generated on AIME25, where we sample 16 responses per question at each training step. Rewarding the simple prefix substantially increases the reasoning step number compared with rewarding the shortened prefix.

To counteract step inflation, we introduce a simple **shorten-prefix** strategy that trims a small suffix of the *good prefix* to assign reward. Let $\mathcal{GP}(q,o) = (o_1,\ldots,o_{t_{M_{\text{err}}}-1})$ and $c(q,o)$ be a nonnegative cut length. The shorten prefix strategy considers $\mathcal{RP}(q,o) := (o_1,\ldots,o_{t_{M_{\text{err}}}-c(q,o)-1})$, which discards the last $c(q,o)$ tokens from the *good prefix*. If $c(q,o) > M_{\text{err}} - 1$, then $\mathcal{GP}_{\text{short}}(q,o) = \emptyset$. Intuitively, this creates an unrewarded “buffer” near the first-error boundary: micro-steps introduced near the tail of the *good prefix* largely fall into the trimmed region and therefore no longer increase the rewarded tokens. This breaks the direct incentive to inflate the number of steps and shifts learning pressure toward improving the quality of earlier reasoning. In practice, we found that set $c(q,o)$ to the token length of the prompt with question q has the best performance, and we use it for Figure 5 and the following results. As shown in Figure 5, shorten prefix substantially reduces the number of reasoning steps in model outputs, and we will show it also increase the performance. More discussion on the choice and effect of $\mathcal{RP}$ is in Appendix G.

4.1 Main Results

Intuitively, as shown in Section 3, our new reward scheme take advantage of learning signals that standard sparse rewards fail to use, thereby improving both exploitation (Pass@1) and exploration (Pass@K).

Pass@1 Results. Table 1 reports Average@16 on several benchmarks, obtained by sampling 16 responses per question and averaging accuracy. Across all settings, our method consistently outperforms both standard GRPO and existing PRM-enhanced reasoning algorithms. We found that the simple Mixed method underperforms GRPO, which is consistent with Ye et al. (2025). For RTS, Chen et al. (2025a) show gains over GRPO on high-capacity distilled and instruction-tuned models; our experiments on Qwen3-4B replicate this advantage on the instruction-tuned models but reveal that RTS falls below GRPO on base models. In contrast, our VPPO (with the shorten-prefix strategy) not only improves over the sparse-reward baseline (GRPO) but also **consistently surpasses both Mixed and RTS on base and instruction-tuned models**, demonstrating the robustness of our approach. Figure 4 further shows test-accuracy training dynamic on AIME 2025, where our method dominates the baselines throughout training.

Pass@K Results. The Pass@K metric measures whether a model can solve a question within K independent attempts, and thus characterizes the model's *reasoning boundary* under repeated sampling. To estimate Pass@K, we utilize the unbiased estimation with low-variance proposed in (Chen, 2021). Specifically, for a given a question q, we draw $n = 256$ samples and if c of them are correct, we calculate $\text{Pass@K}(q) = 1 - \frac{\binom{n-c}{K}}{\binom{n}{K}}$. For a dataset $\mathcal{Q}$, the average Pass@K is defined as $\text{Pass@K}(\mathcal{Q}) = \frac{1}{|\mathcal{Q}|}\sum_{q\in\mathcal{Q}} \text{Pass@K}(q)$. Since we use $n = 256$ samples per question, we report Pass@K only up to $K = 128$ to keep the estimator variance low. The average Pass@K results on dataset AIME24 and AIME25 are summarized in Table 2. We can observe that our method not only has the largest Pass@1, but can increase Pass@K as well compared with GRPO and other PRM-enhanced methods. Notably, our method attains **comparable Pass@K performance** to a dedicated Pass@K optimization algorithm (Chen et al., 2025b) (denote as **PassK** in Table 2) while achieving **much better Pass@1 value**.

Algorithm	**Pass@K**							
	1	2	4	8	16	32	64	128
				Qwen3-4B-Base				
Base	6.5	10.6	15.6	21.2	26.8	33.0	40.1	47.0
GRPO	15.4	20.3	24.6	28.8	33.3	38.3	44.0	49.9
Mixed	12.4	16.9	21.3	25.7	30.2	34.8	40.1	46.5
Ratio	15.6	19.0	21.9	25.5	30.0	34.7	39.3	44.4
PassK	12.2	16.5	20.6	24.9	30.3	37.3	45.8	**54.6**
Ours	**17.4**	**23.0**	**27.8**	**32.3**	**36.9**	**42.1**	**48.0**	53.4
Δ (w. GRPO)	+2.0	+2.7	+3.2	+3.5	+3.6	+3.8	+4.0	+3.5
				Qwen3-8B-Base				
Base	4.4	7.6	12.3	18.0	24.5	31.4	39.3	48.6
GRPO	19.1	23.6	27.8	32.2	36.8	41.4	46.8	53.3
Mixed	16.0	20.1	24.2	29.4	35.4	41.6	47.6	53.6
Ratio	15.9	19.6	23.0	27.3	32.7	38.6	45.1	51.2
PassK	18.8	24.0	**29.4**	**35.3**	**41.4**	**46.5**	**53.0**	56.7
Ours	**20.5**	**24.2**	28.5	33.1	39.0	45.9	52.6	**59.9**
Δ (w. GRPO)	+1.4	+0.6	+0.7	+0.9	+2.2	+4.5	+5.8	+6.6
				Qwen3-4B (non-think)				
Base	18.6	24.2	29.5	35.3	42.0	49.3	56.6	64.1
GRPO	28.1	34.6	41.6	**48.2**	**54.2**	59.5	64.0	68.1
Mixed	20.3	25.3	30.9	37.8	45.7	53.1	59.6	65.3
Ratio	27.2	33.9	40.4	46.7	53.0	59.4	**65.4**	70.2
PassK	25.4	32.3	38.5	44.0	49.8	56.4	63.0	68.8
Ours	**29.5**	**35.9**	**41.7**	47.3	53.4	**59.6**	65.2	**70.6**
Δ (w. GRPO)	+1.4	+1.3	+0.1	-0.9	-0.8	+0.1	+1.2	+2.5

Table 2: **Average Pass@K Values on AIME.** The highest number for each column is bold, and our method improves both Pass@1 and Pass@K.

The impact of different choices of α. In previous discussions, we fix $\alpha = 0.5$. In Appendix H, we test the sensitivity of hyperparameter α, showing that the performance of our algorithm is stable if α is not too large.

5 Conclusions

We present VPPO, a process-supervised RL method that turns noisy step-level PRM scores into a simple, reliable training signal: the location of the first mistake in a sampled reasoning chain. Using this mistake boundary, VPPO preserves and reinforces part of the *good prefix*, the maximal correct segment preceding the first error, via an additional reward, while explicitly penalizing only the erroneous suffix. This reallocation of learning signal fixes core failure mode of outcome-only RL, yielding updates that are both more targeted and stable. Across reasoning benchmarks, VPPO consistently improves Pass@1 and Pass@K results, outperforming sparse-reward baselines and prior PRM-enhanced methods. Future work could further investigate using first-error signals and good prefixes to help learning. For example, during RL, we can store good prefixes in a replay buffer and resample them to train the model to complete from diverse prefixes, which may improve generalization.

6 Limitation

The method prompts the policy to enumerate reasoning as “Step 1…Step 2…”, then defines steps by matching the literal marker. This creates a vulnerability: the model can “inflate” steps (split a large step into many micro-steps) to increase the rewarded token mass, causing overly verbose reasoning. The proposed shorten-prefix trick mitigates this, but the approach still fundamentally depends on this step format and parsing pipeline. The shorten-prefix scheme requires choosing a cut length $c(q)$; the paper’s best-performing choice is setting $c(q)$ to the prompt token length, which is effective empirically but should not be universally optimal. More deep understanding about the step inflaction behaviour and more principled method to tackle it should be further investigated. Moreover, it is beneficial to run experiments on more diverse models.

References

Mislav Balunović, Jasper Dekoninck, Ivo Petrov, Nikola Jovanović, and Martin Vechev. 2025. Matharena: Evaluating llms on uncontaminated math competitions.

Mark Chen. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Peter Chen, Xiaopeng Li, Ziniu Li, Xi Chen, and Tianyi Lin. 2025a. Spectral policy optimization: Coloring your incorrect reasoning in grpo. *arXiv preprint arXiv:2505.11595*.

Zhipeng Chen, Xiaobo Qin, Youbin Wu, Yue Ling, Qinghao Ye, Wayne Xin Zhao, and Guang Shi. 2025b. Pass@ k training for adaptively balancing exploration and exploitation of large reasoning models. *arXiv preprint arXiv:2508.10751*.

Jie Cheng, Gang Xiong, Ruixi Qiao, Lijun Li, Chao Guo, Junle Wang, Yisheng Lv, and Fei-Yue Wang. 2025. Stop summation: Min-form credit assignment is all process reward model needs for reasoning. *arXiv preprint arXiv:2504.15275*.

Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Yuchen Zhang, Jiacheng Chen, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu, and 1 others. 2025a. Process reinforcement through implicit rewards. *arXiv preprint arXiv:2502.01456*.

Ganqu Cui, Yuchen Zhang, Jiacheng Chen, Lifan Yuan, Zhi Wang, Yuxin Zuo, Haozhan Li, Yuchen Fan, Huayu Chen, Weize Chen, and 1 others. 2025b. The entropy mechanism of reinforcement learning for reasoning language models. *arXiv preprint arXiv:2505.22617*.

Runpeng Dai, Linfeng Song, Haolin Liu, Zhenwen Liang, Dian Yu, Haitao Mi, Zhaopeng Tu, Rui Liu, Tong Zheng, Hongtu Zhu, and 1 others. 2025. Cde: Curiosity-driven exploration for efficient reinforcement learning in large language models. *arXiv preprint arXiv:2509.09675*.

Kaituo Feng, Kaixiong Gong, Bohao Li, Zonghao Guo, Yibing Wang, Tianshuo Peng, Junfei Wu, Xiaoying Zhang, Benyou Wang, and Xiangyu Yue. 2025. Video-r1: Reinforcing video reasoning in mllms. *arXiv preprint arXiv:2503.21776*.

Kanishk Gandhi, Ayush Chakravarthy, Anikait Singh, Nathan Lile, and Noah D Goodman. 2025. Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective stars. *arXiv preprint arXiv:2503.01307*.

Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D Goodman. 2024. Stream of search (sos): Learning to search in language. *arXiv preprint arXiv:2404.03683*.

Jiaxuan Gao, Shusheng Xu, Wenjie Ye, Weilin Liu, Chuyi He, Wei Fu, Zhiyu Mei, Guangju Wang, and Yi Wu. 2024. On designing effective rl reward at training time for llm reasoning. *arXiv preprint arXiv:2410.15115*.

Zhaolin Gao, Joongwon Kim, Wen Sun, Thorsten Joachims, Sid Wang, Richard Yuanzhe Pang, and Liang Tan. 2025. Prompt curriculum learning for efficient llm post-training. *arXiv preprint arXiv:2510.01135*.

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

Yiran Guo, Lijie Xu, Jie Liu, Dan Ye, and Shuang Qiu. 2025b. Segment policy optimization: Effective segment-level credit assignment in rl for large language models. *arXiv preprint arXiv:2505.23564*.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng 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. *arXiv preprint arXiv:2402.14008*.

Yicheng He, Chengsong Huang, Zongxia Li, Jiaxin Huang, and Yonghui Yang. 2025. Visplay: Self-evolving vision-language models from images. *arXiv preprint arXiv:2511.15661*.

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

Jian Hu, Xibin Wu, Zilin Zhu, Weixun Wang, Dehao Zhang, Yu Cao, and 1 others. 2024. Openrlhf: An easy-to-use, scalable and high-performance rlhf framework. *arXiv preprint arXiv:2405.11143*.

Zhiyuan Hu, Yibo Wang, Hanze Dong, Yuhui Xu, Amrita Saha, Caiming Xiong, Bryan Hooi, and Junnan Li. 2025. Beyond'aha!': Toward systematic meta-abilities alignment in large reasoning models. *arXiv preprint arXiv:2505.10554*.

Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, and Dong Yu. 2025a. R-zero: Self-evolving reasoning llm from zero data. *arXiv preprint arXiv:2508.05004*.

Chengsong Huang, Tong Zheng, Langlin Huang, Jinyuan Li, Haolin Liu, and Jiaxin Huang. 2026. Relayllm: Efficient reasoning via collaborative decoding. *arXiv preprint arXiv:2601.05167*.

Qihan Huang, Weilong Dai, Jinlong Liu, Wanggui He, Hao Jiang, Mingli Song, Jingyuan Chen, Chang Yao, and Jie Song. 2025b. Boosting mllm reasoning with text-debiased hint-grpo. *arXiv preprint arXiv:2503.23905*.

Szu-chi Huang, Liyin Jin, and Ying Zhang. 2017. Step by step: Sub-goals as a source of motivation. *Organizational Behavior and Human Decision Processes*, 141:1–15.

Wenxuan Huang, Bohan Jia, Zijie Zhai, Shaosheng Cao, Zheyu Ye, Fei Zhao, Zhe Xu, Yao Hu, and Shaohui Lin. 2025c. Vision-r1: Incentivizing reasoning capability in multimodal large language models. *arXiv preprint arXiv:2503.06749*.

Sinan Ibrahim, Mostafa Mostafa, Ali Jnadi, Hadi Salloum, and Pavel Osinenko. 2024. Comprehensive overview of reward engineering and shaping in advancing reinforcement learning applications. *IEEE Access*.

Tom Jurgenson, Or Avner, Edward Groshev, and Aviv Tamar. 2020. Sub-goal trees a framework for goal-based reinforcement learning. In *International conference on machine learning*, pages 5020–5030. PMLR.

Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordoni, Siva Reddy, Aaron Courville, and Nicolas Le Roux. 2024. Vineppo: Refining credit assignment in rl training of llms. *arXiv preprint arXiv:2410.01679*.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, and 1 others. 2024. T\" ulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, and 1 others. 2022. Solving quantitative reasoning problems with language models. *Advances in neural information processing systems*, 35:3843–3857.

Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Huang, Kashif Rasul, Longhui Yu, Albert Q Jiang, Ziju Shen, and 1 others. 2024a. Numinamath: The largest public dataset in ai4maths with 860k pairs of competition math problems and solutions. *Hugging Face repository*, 13(9):9.

Long Li, Xuzheng He, Haozhe Wang, Linlin Wang, and Liang He. 2024b. How do humans write code? large models do it the same way too. *arXiv preprint arXiv:2402.15729*.

Tianjian Li, Yiming Zhang, Ping Yu, Swarnadeep Saha, Daniel Khashabi, Jason Weston, Jack Lanchantin, and Tianlu Wang. 2025a. Jointly reinforcing diversity and quality in language model generations. *arXiv preprint arXiv:2509.02534*.

Wendi Li and Yixuan Li. 2024. Process reward model with q-value rankings. *arXiv preprint arXiv:2410.11287*.

Zongxia Li, Wenhao Yu, Chengsong Huang, Rui Liu, Zhenwen Liang, Fuxiao Liu, Jingxi Che, Dian Yu, Jordan Boyd-Graber, Haitao Mi, and 1 others. 2025b. Self-rewarding vision-language model via reasoning decomposition. *arXiv preprint arXiv:2508.19652*.

Zhenwen Liang, Sidi Lu, Wenhao Yu, Kishan Panaganti, Yujun Zhou, Haitao Mi, and Dong Yu. 2025. Can llms guide their own exploration? gradient-guided reinforcement learning for llm reasoning. *arXiv preprint arXiv:2512.15687*.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *arXiv preprint arXiv:2305.20050*.

Mingjie Liu, Shizhe Diao, Ximing Lu, Jian Hu, Xin Dong, Yejin Choi, Jan Kautz, and Yi Dong. 2025a. Prorl: Prolonged reinforcement learning expands reasoning boundaries in large language models. *arXiv preprint arXiv:2505.24864*.

Mingyang Liu, Gabriele Farina, and Asuman Ozdaglar. 2025b. Uft: Unifying supervised and reinforcement fine-tuning. *arXiv preprint arXiv:2505.16984*.

Rui Liu, Dian Yu, Lei Ke, Haolin Liu, Yujun Zhou, Zhenwen Liang, Haitao Mi, Pratap Tokekar, and Dong Yu. 2025c. Stable and efficient single-rollout rl for multimodal reasoning. *arXiv preprint arXiv:2512.18215*.

Rui Liu, Dian Yu, Tong Zheng, Runpeng Dai, Zongxia Li, Wenhao Yu, Zhenwen Liang, Linfeng Song, Haitao Mi, Pratap Tokekar, and 1 others. 2025d. Vogue: Guiding exploration with visual uncertainty improves multimodal reasoning. *arXiv preprint arXiv:2510.01444*.

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

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, Jiao Sun, and 1 others. 2024. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592*.

Trung Quoc Luong, Xinbo Zhang, Zhanming Jie, Peng Sun, Xiaoran Jin, and Hang Li. 2024. Reft: Reasoning with reinforced fine-tuning. *arXiv preprint arXiv:2401.08967*.

Zexiong Ma, Chao Peng, Pengfei Gao, Xiangxin Meng, Yanzhen Zou, and Bing Xie. 2025. Sorft: Issue resolving with subtask-oriented reinforced fine-tuning. *arXiv preprint arXiv:2502.20127*.

Tong Mu, Alec Helyar, Johannes Heidecke, Joshua Achiam, Andrea Vallone, Ian Kivlichan, Molly Lin, Alex Beutel, John Schulman, and Lilian Weng. 2024. Rule based rewards for language model safety. *arXiv preprint arXiv:2411.01111*.

Andrew Y Ng, Daishi Harada, and Stuart Russell. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287. Citeseer.

Takato Okudo and Seiji Yamada. 2021. Subgoal-based reward shaping to improve efficiency in reinforcement learning. *IEEE Access*, 9:97557–97568.

Sujoy Paul, Jeroen Vanbaar, and Amit Roy-Chowdhury. 2019. Learning from trajectories via subgoal discovery. *Advances in Neural Information Processing Systems*, 32.

Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. 2024. Rewarding progress: Scaling automated process verifiers for llm reasoning. *arXiv preprint arXiv:2410.08146*.

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*.

Haozhan Shen, Peng Liu, Jingcheng Li, Chunxin Fang, Yibo Ma, Jiajia Liao, Qiaoli Shen, Zilun Zhang, Kangjia Zhao, Qianqian Zhang, and 1 others. 2025. Vlm-r1: A stable and generalizable r1-style large vision-language model. *arXiv preprint arXiv:2504.07615*.

Taiwei Shi, Yiyang Wu, Linxin Song, Tianyi Zhou, and Jieyu Zhao. 2025. Efficient reinforcement finetuning via adaptive curriculum learning. *arXiv preprint arXiv:2504.05520*.

Mingyang Song, Zhaochen Su, Xiaoye Qu, Jiawei Zhou, and Yu Cheng. 2025a. Prmbench: A fine-grained and challenging benchmark for process-level reward models. *arXiv preprint arXiv:2501.03124*.

Yuda Song, Julia Kempe, and Remi Munos. 2025b. Outcome-based exploration for llm reasoning. *arXiv preprint arXiv:2509.06941*.

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*.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2024. 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)*, pages 9426–9439.

Liang Wen, Yunke Cai, Fenrui Xiao, Xin He, Qi An, Zhenyu Duan, Yimin Du, Junchen Liu, Tanglifu Tanglifu, Xiaowei Lv, and 1 others. 2025. Light-r1: Curriculum sft, dpo and rl for long cot from scratch and beyond. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pages 318–327.

Tian Xie, Zitian Gao, Qingnan Ren, Haoming Luo, Yuqian Hong, Bryan Dai, Joey Zhou, Kai Qiu, Zhirong Wu, and Chong Luo. 2025. Logic-rl: Unleashing llm reasoning with rule-based reinforcement learning. *arXiv preprint arXiv:2502.14768*.

Wei Xiong, Wenting Zhao, Weizhe Yuan, Olga Golovneva, Tong Zhang, Jason Weston, and Sainbayar Sukhbaatar. 2025. Stepwiser: Stepwise generative judges for wiser reasoning. *arXiv preprint arXiv:2508.19229*.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, and 1 others. 2024. Qwen2. 5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*.

Zhaohui Yang, Chenghua He, Xiaowen Shi, Linjing Li, Qiyue Yin, Shihong Deng, and Daxin Jiang. 2025. Beyond the first error: Process reward models for reflective mathematical reasoning. *arXiv preprint arXiv:2505.14391*.

Chenlu Ye, Zhou Yu, Ziji Zhang, Hao Chen, Narayanan Sadagopan, Jing Huang, Tong Zhang, and Anurag Beniwal. 2025. Beyond correctness: Harmonizing process and outcome rewards through rl training. *arXiv preprint arXiv:2509.03403*.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, and 1 others. 2025a. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*.

Wenhao Yu, Zhenwen Liang, Chengsong Huang, Kishan Panaganti, Tianqing Fang, Haitao Mi, and Dong Yu. 2025b. Guided self-evolving llms with minimal human supervision. *arXiv preprint arXiv:2512.02472*.

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

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*.

Kaiwen Zha, Zhengqi Gao, Maohao Shen, Zhang-Wei Hong, Duane S Boning, and Dina Katabi. 2025. Rl tango: Reinforcing generator and verifier together for language reasoning. *arXiv preprint arXiv:2505.15034*.

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*.

Yuxiang Zhang, Yuqi Yang, Jiangming Shu, Yuhang Wang, Jinlin Xiao, and Jitao Sang. 2024. Openrft: Adapting reasoning foundation model for domain-specific tasks with reinforcement fine-tuning. *arXiv preprint arXiv:2412.16849*.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2025b. The lessons of developing process reward models in mathematical reasoning. *arXiv preprint arXiv:2501.07301*.

Yulai Zhao, Haolin Liu, Dian Yu, Sunyuan Kung, Meijia Chen, Haitao Mi, and Dong Yu. 2025. One token to fool llm-as-a-judge. *arXiv preprint arXiv:2507.08794*.

Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2024. Processbench: Identifying process errors in mathematical reasoning. *arXiv preprint arXiv:2412.06559*.

Congmin Zheng, Jiachen Zhu, Jianghao Lin, Xinyi Dai, Yong Yu, Weinan Zhang, and Mengyue Yang. 2025a. Cold: Counterfactually-guided length debiasing for process reward models. *arXiv preprint arXiv:2507.15698*.

Tong Zheng, Hongming Zhang, Wenhao Yu, Xiaoyang Wang, Xinyu Yang, Runpeng Dai, Rui Liu, Huiwen Bao, Chengsong Huang, Heng Huang, and 1 others. 2025b. Parallel-r1: Towards parallel thinking via reinforcement learning. *arXiv preprint arXiv:2509.07980*.

Yujun Zhou, Zhenwen Liang, Haolin Liu, Wenhao Yu, Kishan Panaganti, Linfeng Song, Dian Yu, Xiangliang Zhang, Haitao Mi, and Dong Yu. 2025. Evolving language models without labels: Majority drives selection, novelty promotes variation. *arXiv preprint arXiv:2509.15194*.

Xinyu Zhu, Mengzhou Xia, Zhepei Wei, Wei-Lin Chen, Danqi Chen, and Yu Meng. 2025. The surprising effectiveness of negative reinforcement in llm reasoning. *arXiv preprint arXiv:2506.01347*.

Jiaru Zou, Ling Yang, Jingwen Gu, Jiahao Qiu, Ke Shen, Jingrui He, and Mengdi Wang. 2025. Reasonflux-prm: Trajectory-aware prms for long chain-of-thought reasoning in llms. *arXiv preprint arXiv:2506.18896*.

A Related Works

Reinforcement learning for LLM reasoning with verifiable rewards. In many LLM reasoning tasks, the final answer can be readily extracted and checked against a reference using either rule-based verifiers (Mu et al., 2024) or LLM-as-a-judge protocols (Luong et al., 2024; Li et al., 2024b; Ma et al., 2025; Xie et al., 2025; Zhao et al., 2025). This outcome signal naturally serves as a reward for reinforcement learning, and this paradigm has proven highly effective for improving reasoning performance across domains (Lambert et al., 2024; Gandhi et al., 2024; Zhang et al., 2024; Guo et al., 2025a; Team et al., 2025). Similar ideas have also been extended to multi-modal reasoning (Huang et al., 2025c; Li et al., 2025b; Shen et al., 2025; Liu et al., 2025d,c; Feng et al., 2025).

Despite these successes, standard RL algorithms face several fundamental challenges in this setting. First, insufficient exploration can cause policy entropy to collapse rapidly, which may even shrink the model's reasoning boundary after RL (Cui et al., 2025b; Yue et al., 2025). To mitigate this issue, a line of work augments the reward with a novelty bonus (often applied at the final token), assigning higher rewards to more unique or uncertain responses to encourage diversity. Empirically, such bonuses can expand policy coverage and further improve performance (Yu et al., 2025a; Cui et al., 2025b; Liu et al., 2025a; Zhu et al., 2025; Li et al., 2025a; Dai et al., 2025; Song et al., 2025b; Zhou et al., 2025; Liang et al., 2025). Second, learning signals can be extremely sparse on difficult prompts: when correct solutions are rarely sampled, rewards are frequently zero, yielding weak or no gradients and slowing progress on hard questions. Prior work addresses this in multiple ways, including adding a mid-training stage that instills meta-behaviors such as reflection or parallel thinking to boost high-level reasoning capability and increase the likelihood of generating correct attempts (Gandhi et al., 2025; Hu et al., 2025; Zheng et al., 2025b; Huang et al., 2026); and using curriculum learning to keep the training distribution near an intermediate difficulty regime where gradient signal is strongest (Wen et al., 2025; Shi et al., 2025; Huang et al., 2025a; He et al., 2025; Gao et al., 2025; Yu et al., 2025b). In our approach, we also address both exploration and signal sparsity by explicitly preserving and learning from the correct prefix of incorrect responses.

Process Supervision for Reasoning. Process supervision has been widely used in reinforcement learning to provide dense learning signals at intermediate steps. In classic RL domains such as games and robotics, these signals are often straightforward to design based on measurable task progress, and dense rewards can substantially accelerate learning (Ng et al., 1999; Huang et al., 2017; Paul et al., 2019; Jurgenson et al., 2020; Okudo and Yamada, 2021; Ibrahim et al., 2024). In contrast, for LLM reasoning, progress at each step is difficult to formalize, motivating the use of learned process reward models (PRMs) that score intermediate reasoning steps (Lightman et al., 2023). Since their introduction, PRMs have primarily been used to improve test-time scaling (Lightman et al., 2023; Wang et al., 2024; Yang et al., 2024; Luo et al., 2024; Yuan et al., 2024; Li and Li, 2024; Yang et al., 2025; Xiong et al., 2025). However, multiple works have found it challenging to integrate PRM signals into reinforcement learning due to substantial score bias and susceptibility to reward hacking (Setlur et al., 2024; Gao et al., 2024; Cheng et al., 2025; Ye et al., 2025). Existing successful attempts often adopt actor-critic style frameworks in which the PRM (as a critic) is trained and updated jointly to track the evolving policy model (Setlur et al., 2024; Kazemnejad et al., 2024; Guo et al., 2025b; Cui et al., 2025a; Zha et al., 2025). While effective, these approaches introduce additional optimization and memory overhead for a large critic model, and may require extra rollouts from intermediate states to improve value estimation, increasing training time significantly. Our approach instead studies how to leverage a well-trained PRM that is fixed during training, avoiding the optimization and memory overhead of jointly learning a critic model. To mitigate reward hacking, rather than directly optimizing uncalibrated per-step PRM scores, we use the PRM scores to localize the first incorrect step in a reasoning trace. This produces a discrete supervision signal with an unambiguous ground truth. Moreover, the localization accuracy can be assessed with dedicated benchmarks (Zheng et al., 2024; Song et al., 2025a), which enables offline threshold selection for each PRM to maximize error-detection reliability. Using this first-error signal effectively, our method outperforms prior approaches that apply a fixed PRM during reinforcement learning training (Zou et al., 2025; Chen et al., 2025a).

B Experiment Details about Figure 1

To get Figure 1, we collect all incorrect rollouts with step-enumeration pattern during the training process of Qwen3-4B-Base in the first three epoch and randomly select 3000 samples. To find the first incorrect step, we utilize Qwen2.5-Math-PRM-7B which outputs a real-valued score per step. Since our objective is to detect the first incorrect step, we convert step scores into binary correctness labels threshold 0.8: the first step whose PRM score falls below 0.8 is treated as the first incorrect step. As shown in Table 5, this threshold can reliably detect the first incorrect step. To further validate the detection accuracy, we randomly sample 300 responses and check whether the first error found by PRM matches the result of GPT-o1, with prompt given in the end of Appendix B. The results are given in Table 3, in which we exclude 6 samples where the output of GPT-o1 is not successfully parsed. From Table 3, we know that PRM estimation is more conservative compared to GPT-o1, indicating that the length of good prefix is probably even more than our evaluation.

Table 3: **Accuracy on PRM compared with GPT-o1.** “Match”, “Less”, and “More” indicate that the predicted index of the first incorrect step is equal to, smaller than, and larger than the predicted incorrect step of GPT-o1, respectively. “Fail” means the PRM does not detect any incorrect step.

Type	Match	Less	More	Fail
Ratio(%)	60.3	25.0	9.3	3.3

To probe how useful these correct prefixes are, Figure 6 analyzes AIME25 that contains 30 questions using Qwen3-4B-Base trained after 3 epochs. For each question, we first sample 32 full responses to compute standard Pass@32, then focus on incorrect responses and extract *good prefixes*, defined as the sequence of correct reasoning steps up to (but not including) the first error. We define *Prefix Pass@32* for a question to be 1 if there exists at least one such good prefix for which, when we restart from this prefix and sample 32 continuations, at least one continuation yields the correct final answer; otherwise it is 0. Empirically, we observe that eight questions have Pass@32 = 0 while Prefix Pass@32 = 1, none of the original 32 full responses solve the problem, yet some continuation from a good prefix does, whereas only a single question has Pass@32 = 1 but Prefix Pass@32 = 0. These results show that good prefixes in incorrect samples can be systematically exploited to obtain correct solutions that naive outcome-only sampling fails to produce, and that such intermediate supervision is largely ignored by standard RL objectives.

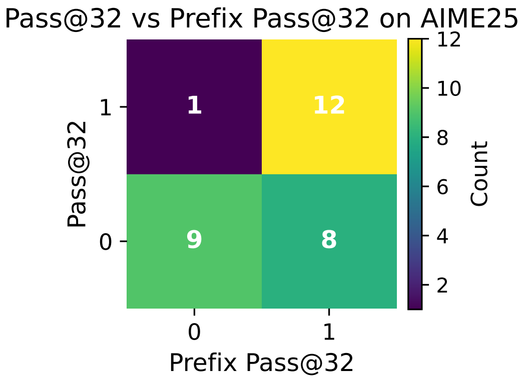


Figure 6: **Good Steps in Incorrect Responses Can Lead to Correct Answers.** We compare Pass@32 with *Prefix Pass@32*, which is 1 if any correct prefix from an incorrect response yields at least one correct continuation among 32 samples. Eight questions have Pass@32 = 0 but Prefix Pass@32 = 1, while only one shows the opposite pattern.

Prompt for GPT-o1 to find the first incorrect step

Given a question with its ground-truth answer and a reasoning that lead to incorrect answer, determine which step the reasoning process makes the first mistake.

Your output should be "Step X" where X is the step number (for example, if the third step in the reasoning process is the first mistake, your output should be "Step 3").

Question:
{question}

Ground-truth answer:
{reference}

Incorrect reasoning process:
{response}
Output:

C Implementation Details

Algorithm in (Zou et al., 2025). They utilize a process reward model which can output the score of every step. For a given reasoning trajectory $(x, y_1, \cdots, y_H)$ where $y_{1:H}$ denote H steps in the reasoning path and let $y_{<h} = (x, y_1, \cdots, y_{h-1})$. (Zou et al., 2025) utilizes a process reward model r to score every step and get $r(y_{<h}, y_h)$ for every $h \in [H]$. Together with the final correctness reward r_{corr}, the final reward of a response y for prompt x is defined as $r_{\text{final}}(x, y) = \frac{\beta}{H}\sum_{h=1}^{H} r(y_{<h}, y_h) + (1-\beta) r_{\text{corr}}(x, y)$. This reward is then used to compute the standard GRPO advantage and utilize loss Eq. (3) for policy update. We choosde $\beta = 0.8$ which has the best performance according to (Zou et al., 2025).

Algorithm in (Chen et al., 2025a). They use a large external models to find the first incorrect step in a incorrect step, assuming a response y has $s(y)$ steps and the first $m(y)$ steps are correct, this method define $\mathrm{RTS}(y) = \frac{m(y)}{s(y)}$ and for incorrect sample, the reward is defined as $\frac{1}{1+\exp(\beta \mathrm{RTS}(y)+\gamma)}$ where β and γ are two hyperparameters[2]. For correct response, the reward is always 1. Using this reward scheme, they also the compute standard GRPO advantage and utilize loss Eq. (3) for policy update.

We utilize OpenRLHF(Hu et al., 2024) to implement our algorithm and the hyperparameters are shown in Table 4. For Qwen3-4B-Base and Qwen3-4B, every training spend roughly 384 GPU-hours. For Qwen3-8B-Base, every training spend 768 GPU-hours.

Table 4: Hyperparameters used in OpenRLHF training.

Hyperparameter	**Value**
micro_train_batch_size	1
train_batch_size	256
micro_rollout_batch_size	8
rollout_batch_size	128
n_samples_per_prompt	8
generate_max_len	4096
actor_learning_rate	5×10^{-7}
init_kl_coef	0
eps_clip	0.2
temperature	1.0
num_episodes	5

For evaluation, we set the sample temperature to be 0.6.

D Implementation Details

D.1 Prompts

We use prompt

System Prompt

You are a helpful assistant, Please solve the problem step by step and follow this layout:
Step 1: ...
(blank line)
Step 2: ...
(blank line)
...
At the end, output the final answer using the exact format: \boxed{your_answer}.

D.2 PRM Score on ProcessBench

We use Qwen2.5-7B-MATH-PRM to identify the first incorrect step in a given response. Since Qwen2.5-7B-MATH-PRM outputs a score for every step, we need to select a threshold and define the predicted first incorrect step as the first step whose score falls below this threshold. To select a suitable threshold, we evaluate the model on ProcessBench (Zheng et al., 2024), which provides reasoning trajectories annotated with the ground-truth index of the first incorrect step.

During training, we only query the PRM for responses whose final answer is incorrect. Accordingly, for our analysis on ProcessBench, we restrict attention to trajectories whose final answer is incorrect, yielding 1700 samples. On this subset, we measure the performance of Qwen2.5-7B-MATH-PRM under different thresholds; the results are reported in Table 5. We observe that Qwen2.5-7B-MATH-PRM tends to assign relatively high scores, so larger thresholds (up to 0.8) lead to higher accuracy in detecting the first incorrect step.

Our primary goal is to reward the correct prefix of the reasoning process. Consequently, it is benign if the PRM predicts the first incorrect step *earlier* than the true one (i.e., a smaller step index): in this case we still only reward steps that are in fact correct. In contrast, if the PRM predicts the first incorrect step *later* than the true one (i.e., a larger step index), then some truly incorrect steps would be mistakenly rewarded, which can hinder policy optimization. Another safe failure mode is when the PRM does not identify any incorrect step for an incorrect response; in this case we assign zero

---

[2] The authors do not explicitly mention their chosen value in their paper and we ask them through email to know that they choose $\beta = -10$ and $\gamma = 20$

reward to all tokens, ensuring that the incorrect step is not positively reinforced.

Therefore, the only harmful misspecification for learning is when the predicted first incorrect step index is *larger* than the ground-truth index. This corresponds to the “More” row in Table 5. The last row, “Not More”, reports the probability that the predicted index is not larger than the ground truth (including exact matches, earlier detections, and failures to detect any incorrect step), i.e., the probability that the additional PRM signal is non-harmful. We see that, with an appropriately chosen threshold, in over 90% of the cases the PRM provides a reliable signal. Balancing overall accuracy and the Not More rate, we set the threshold to 0.8 in all our experiments.

Table 5: **Accuracy on ProcessBench for different PRM thresholds.** “Match”, “Less”, and “More” indicate that the predicted index of the first incorrect step is equal to, smaller than, and larger than the ground-truth index, respectively. “Fail” means the PRM does not detect any incorrect step. “Not More” is the proportion of cases where the predicted index is not larger than the ground truth, i.e., where the PRM signal is non-harmful. All entries are ratios (%).

Threshold	0.5	0.6	0.7	0.8	0.9
Match	61.3	63.7	63.1	**63.2**	58.5
Less	11.2	14.9	20.1	25.0	34.4
More	14.6	12.0	10.2	7.3	4.5
Fail	12.9	9.5	6.6	4.5	2.7
Not More	85.4	88.0	89.8	92.7	95.5

E Case Studies

E.1 Exploration Example

In this section, we present a case study corresponding to the exploration example in Section 3. For the same question, we observe two rollouts: one correct and one incorrect, which follow different reasoning paths. The incorrect rollout contains several correct steps before making its first mistake; the first incorrect step, as identified by the PRM, is highlighted in red.

Question

Find the value of $L = \lim_{x \to 0} \frac{e^x - \cos x - x}{x^2}$.

Rollout 1 (Correct)

To find the value of $L = \lim_{x \to 0} \frac{e^x - \cos x - x}{x^2}$, we will use L’Hopital’s rule. L’Hopital’s rule states that if the limit of the quotient of two functions results in an indeterminate form such as $\frac{0}{0}$ or $\frac{\infty}{\infty}$, then the limit of the quotient is equal to the limit of the quotient of their derivatives.
Step 1: Evaluate the limit directly. Let’s plug in $x = 0$:

$$\frac{e^0 - \cos 0 - 0}{0^2} = \frac{1 - 1}{0} = \frac{0}{0}.$$

This is an indeterminate form, so we can apply L’Hopital’s rule.
Step 2: Apply L’Hopital’s rule. We need to differentiate the numerator and the denominator separately and then take the limit again. The derivative of the numerator, $e^x - \cos x - x$, is:

$$\frac{d}{dx}(e^x - \cos x - x) = e^x + \sin x - 1.$$

The derivative of the denominator, x^2, is:

$$\frac{d}{dx}(x^2) = 2x.$$

So the limit becomes:

$$L = \lim_{x \to 0} \frac{e^x + \sin x - 1}{2x}.$$

Step 3: Evaluate the new limit. Now let’s plug in $x = 0$:

$$\frac{e^0 + \sin 0 - 1}{2 \cdot 0} = \frac{1 + 0 - 1}{0} = \frac{0}{0}.$$

Again, we have an indeterminate form, so we need to apply L’Hopital’s rule once more.
Step 4: Apply L’Hopital’s rule again. Differentiate the numerator and the denominator again: The derivative of the numerator, $e^x + \sin x - 1$, is:

$$\frac{d}{dx}(e^x + \sin x - 1) = e^x + \cos x.$$

The derivative of the denominator, $2x$, is:

$$\frac{d}{dx}(2x) = 2.$$

So the limit now is:

$$L = \lim_{x \to 0} \frac{e^x + \cos x}{2}.$$

Step 5: Evaluate the limit. Now let's plug in $x = 0$:

$$\frac{e^0 + \cos 0}{2} = \frac{1+1}{2} = \frac{2}{2} = 1.$$

Therefore, the value of L is:

$$\boxed{1}.$$

Rollout 2 (InCorrect)

To find the value of $L = \lim_{x \to 0} \frac{e^x - \cos x - x}{x^2}$, we can use Taylor series expansion.
Step 1: Recall the Taylor series expansions for e^x and $\cos x$ around $x = 0$. Step 1: Recall the Taylor series expansions for e^x and $\cos x$ around $x = 0$.
Step 2: The Taylor series expansion of e^x around $x = 0$ is:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

.
Step 3: The Taylor series expansion of $\cos x$ around $x = 0$ is:

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \cdots$$

.
Step 4: Substitute these expansions into the limit expression:

$$L = \lim_{x \to 0} \frac{\left(1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\cdots\right)-\left(1-\frac{x^2}{2!}+\frac{x^4}{4!}-\cdots\right)-x}{x^2}$$

Step 5: Simplify the expression by combining like terms:

$$L = \lim_{x \to 0} \frac{x^2 + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots}{x^2}$$

Step 6: Cancel out all terms except the one containing x^2:

$$L = \lim_{x \to 0} \left(1 + \frac{1}{2!} + \frac{x}{3!} + \cdots\right)$$

Step 7: Take the limit as x approaches 0:

$$L = 1 + \frac{1}{2!} = 1 + \frac{1}{2} = \boxed{\frac{3}{2}}$$

E.2 Exploitation Example

In this section, we present a case study for the exploitation example in Section 3. For the same question, we observe two rollouts, one correct and one incorrect, that share similar initial steps but diverge midway through the reasoning. The incorrect rollout contains several correct steps that closely match those in the correct rollout before its first mistake; this first incorrect step, as identified by the PRM, is highlighted in red.

Question

Let $p(x)$ be the second degree polynomial such that $p(1) = 1$, $p(2) = 3$, and $p(3) = 2$. Then $p(p(x)) = x$ has four real solutions. Find the only such solution which is not an integer.

Rollout 1 (Correct)

To solve for the polynomial $p(x)$ of the second degree such that $p(1) = 1$, $p(2) = 3$, and $p(3) = 2$, we start by expressing $p(x)$ in the general form of a quadratic polynomial:

$$p(x) = ax^2 + bx + c.$$

We need to determine the coefficients a, b, and c using the given conditions.
Step 1: Set up the system of equations From the given conditions, we have: 1. $p(1) = a(1)^2 + b(1) + c = a + b + c = 1$ 2. $p(2) = a(2)^2 + b(2) + c = 4a + 2b + c = 3$ 3. $p(3) = a(3)^2 + b(3) + c = 9a + 3b + c = 2$ This gives us the following system of linear equations:

$$\begin{cases} a + b + c = 1 \\ 4a + 2b + c = 3 \\ 9a + 3b + c = 2 \end{cases}$$

Step 2: Solve the system of equations First, subtract the first equation from the second equation:

$$(4a + 2b + c) - (a + b + c) = 3 - 1$$

$$3a + b = 2 \quad \text{(Equation 4)}$$

Next, subtract the second equation from the third equation:

$$(9a + 3b + c) - (4a + 2b + c) = 2 - 3$$

$$5a + b = -1 \quad \text{(Equation 5)}$$

Now, we have a new system of two equations with two variables:

$$\begin{cases} 3a + b = 2 \\ 5a + b = -1 \end{cases}$$

Subtract Equation 4 from Equation 5:

$$(5a + b) - (3a + b) = -1 - 2$$

$$2a = -3$$

$$a = -\frac{3}{2}$$

Step 3: Find b Substitute $a = -\frac{3}{2}$ back into Equation 4:

$$3\left(-\frac{3}{2}\right) + b = 2$$

$$-\frac{9}{2} + b = 2$$

$$b = 2 + \frac{9}{2}$$

$$b = \frac{4}{2} + \frac{9}{2}$$

$$b = \frac{13}{2}$$

Step 4: Find c Substitute $a = -\frac{3}{2}$ and $b = \frac{13}{2}$ back into the first equation:

$$-\frac{3}{2} + \frac{13}{2} + c = 1$$

$$\frac{10}{2} + c = 1$$

$$5 + c = 1$$

$$c = 1 - 5$$

$$c = -4$$

Step 5: Write the polynomial $p(x)$ Now we have the coefficients:

$$a = -\frac{3}{2}, \quad b = \frac{13}{2}, \quad c = -4$$

Thus, the polynomial is:

$$p(x) = -\frac{3}{2}x^2 + \frac{13}{2}x - 4$$

Step 6: Verify $p(p(x)) = x$ We need to find $p(p(x))$ and set it equal to x:

$$p(p(x)) = p\left(-\frac{3}{2}x^2 + \frac{13}{2}x - 4\right)$$

Let $y = p(x) = -\frac{3}{2}x^2 + \frac{13}{2}x - 4$. Then:

$$p(y) = -\frac{3}{2}y^2 + \frac{13}{2}y - 4$$

Substitute $y = -\frac{3}{2}x^2 + \frac{13}{2}x - 4$ into $p(y)$:

$$\begin{aligned} &p\left(-\frac{3}{2}x^2 + \frac{13}{2}x - 4\right) \\ &= -\frac{3}{2}\left(-\frac{3}{2}x^2 + \frac{13}{2}x - 4\right)^2 \\ &\quad + \frac{13}{2}\left(-\frac{3}{2}x^2 + \frac{13}{2}x - 4\right) - 4 \end{aligned}$$

This is quite complex, so let's instead solve $p(p(x)) = x$ directly by finding the roots of the polynomial equation.

Step 7: Solve $p(p(x)) = x$ We need to find the roots of the equation $p(p(x)) = x$. This is a quartic equation, and we know it has four real solutions. We are asked to find the non-integer solution. Given the complexity, we can use numerical methods or graphing to find the roots. However, we can also use the fact that the polynomial $p(x)$ is quadratic and symmetric around its vertex.

Step 8: Find the roots We can use the quadratic formula to find the roots of $p(x) = x$:

$$-\frac{3}{2}x^2 + \frac{13}{2}x - 4 = x$$

$$-\frac{3}{2}x^2 + \frac{11}{2}x - 4 = 0$$

Multiply through by 2 to clear the fractions:

$$-3x^2 + 11x - 8 = 0$$

$$3x^2 - 11x + 8 = 0$$

Using the quadratic formula $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$:

$$x = \frac{11 \pm \sqrt{121 - 96}}{6}$$

$$x = \frac{11 \pm \sqrt{25}}{6}$$

$$x = \frac{11 \pm 5}{6}$$

This gives us two solutions:

$$x = \frac{16}{6} = \frac{8}{3}$$

$$x = \frac{6}{6} = 1$$

The non-integer solution is:

$$\boxed{\frac{8}{3}}$$

Rollout 2 (InCorrect)

To solve for the polynomial $p(x)$ of the second degree such that $p(1) = 1$, $p(2) = 3$, and $p(3) = 2$, we start by expressing $p(x)$ in the general form of a quadratic polynomial:

$$p(x) = ax^2 + bx + c.$$

We need to determine the coefficients a, b, and c using the given conditions.

Step 1: Set up the system of equations From the given conditions, we have: 1. $p(1) = a(1)^2 + b(1) + c = a + b + c = 1$ 2. $p(2) = a(2)^2 + b(2) + c = 4a + 2b + c = 3$ 3. $p(3) = a(3)^2 + b(3) + c = 9a + 3b + c = 2$ This gives us the following system of linear equations:

$$\begin{cases} a + b + c = 1 \\ 4a + 2b + c = 3 \\ 9a + 3b + c = 2 \end{cases}$$

Step 2: Solve the system of equations First, subtract the first equation from the second equation:

$$(4a + 2b + c) - (a + b + c) = 3 - 1$$

$$3a + b = 2 \quad \text{(Equation 4)}$$

Next, subtract the second equation from the third equation:

$$(9a + 3b + c) - (4a + 2b + c) = 2 - 3$$

$$5a + b = -1 \quad \text{(Equation 5)}$$

Now, we have a new system of two equations with two variables:

$$\begin{cases} 3a + b = 2 \\ 5a + b = -1 \end{cases}$$

Subtract Equation 4 from Equation 5:

$$(5a + b) - (3a + b) = -1 - 2$$

$$2a = -3$$

$$a = -\frac{3}{2}$$

Step 3: Find b Substitute $a = -\frac{3}{2}$ into equation 4:

$$3\left(-\frac{3}{2}\right) + b = 2$$

$$-\frac{9}{2} + b = 2$$

$$b = 2 + \frac{9}{2}$$

$$b = \frac{4}{2} + \frac{9}{2}$$

$$b = \frac{13}{2}$$

Step 4: Find c Substitute $a = -\frac{3}{2}$ and $b = \frac{13}{2}$ into the first equation:

$$-\frac{3}{2} + \frac{13}{2} + c = 1$$

$$\frac{10}{2} + c = 1$$

$$5 + c = 1$$

$$c = 1 - 5$$

$$c = -4$$

Step 5: Write the polynomial $p(x)$

$$p(x) = -\frac{3}{2}x^2 + \frac{13}{2}x - 4.$$

Step 6: Verify $p(p(x)) = x$ We need to find $p(p(x))$ and set it equal to x:

$$p(p(x)) = p\left(-\frac{3}{2}x^2 + \frac{13}{2}x - 4\right).$$

Let $y = -\frac{3}{2}x^2 + \frac{13}{2}x - 4$. Then:

$$p(y) = -\frac{3}{2}y^2 + \frac{13}{2}y - 4.$$

Substitute y back:

$$\begin{aligned}
&p\left(-\frac{3}{2}x^2+\frac{13}{2}x-4\right)\\
&=-\frac{3}{2}\left(-\frac{3}{2}x^2+\frac{13}{2}x-4\right)^2\\
&\qquad+\frac{13}{2}\left(-\frac{3}{2}x^2+\frac{13}{2}x-4\right)-4.
\end{aligned}$$

This is complex, so we use the fact that $p(x)$ is a quadratic polynomial and $p(p(x)) = x$ implies that $p(x)$ is its own inverse up to some transformation. We need to find the roots of $p(p(x)) - x = 0$.

Step 7: Solve $p(p(x)) = x$ Given the complexity, we can use numerical methods or graphing to find the roots. However, we know that $p(x)$ is a quadratic polynomial, and $p(p(x)) = x$ will have four real solutions. We need to find the non-integer solution. By solving the polynomial equations or using numerical methods, we find that the non-integer solution is:

$$\boxed{\frac{5}{3}}$$

F Proof of Theorem 1

We consider a tree-structure Markov Decision Process with layer H as a simplified model for reasoning task. The question q is the root node, and every node denote an action. For any action sequence $a_{<h} = (a_1, \cdots, a_{h-1})$ till step h, the current state is $[q, a_{<h}]$ defined as the concatenation of the question and observed actions so far. Given a randomized policy π, the learner chooses $a_h \sim \pi(\cdot|[q, a_{<h}])$, and the state is deterministically transit to $[q, a_{<h+1}] = [q, a_{<h}, a_h]$. We assume at every state, there are only possible two actions and there is a unique correct trajectory $[a_1^\star, a_2^\star, \cdots, a_H^\star]$ that leads to the correct final answer. We restate Theorem 1 as follows.

Theorem 1. Given a reasoning tree defined above, using standard policy optimization algorithm with learning rate η and uniform initailization, for any $\epsilon \in (0, \frac{1}{2})$, let $N^\star_{\text{sparse}}(\epsilon)$ and $N^\star_{\text{dense}}(\epsilon)$ be the minimum sample size to learn an ϵ-optimal policy under sparse reward scheme Eq. (1) and our reward scheme Eq. (6) respectively, where an ϵ-optimal policy π ensures $\mathbb{P}_{o\sim\pi}\left[o \text{ is correct}\right] \geq 1-\epsilon$. We have

$$\mathbb{E}\left[N^\star_{\text{sparse}}(\epsilon)\right] \geq \Omega\left(\frac{2^H}{\eta H}\right) \text{ and } \mathbb{E}\left[N^\star_{\text{dense}}(\epsilon)\right] \leq \mathcal{O}\left(\frac{H}{\eta\alpha}\log\frac{H}{\epsilon}\right)$$

where Ω and $\mathcal{O}$ only hide universal constants.

Proof. For a given question q as the root, we take a sample at every round and get a reasoning chain $o = (a_1, \cdots, a_H)$, we consider standard policy update with KL regularization such that for any episode t, any layer h and any previous action sequence $o = (a_1, \cdots, a_{h-1})$, we have

$$\pi_{t+1}(a_h|[q, a_{<h}]) = \underset{\pi}{\text{argmax}}\left\{\left\langle \pi, \widehat{Q}_t\left([q, a_{<h}], a_h\right)\right\rangle - \frac{1}{\eta}\text{KL}\left(\pi\left(\cdot|[q, a_{<h}]\right), \pi_t\left(\cdot|[q, a_{<h}]\right)\right)\right\}$$

which has closed-form solution

$$\pi_{t+1}(a_h|[q, a_{<h}]) \propto \pi_t(a_h|[q, a_{<h}]) \exp\left(\eta \widehat{Q}_t([q, a_{<h}], a_h)\right) \tag{9}$$

where we set $\widehat{Q}_t([q, a_{<h}], a_h)$ to be the standard Monte-Carlo estimation for prefix $[q, a_{<h}]$ and step a_h. Specifically, assume we have a rollout $[q, a_1^t, \cdots, a_H^t]$ at time t, if the last action a_t^H has correct answer, then $\widehat{Q}_t([q, a_{<h}^t], a_h^t) = 1$ for every $h \in [H]$, and $\widehat{Q}_t([q, a'_{<h}], a'_h) = 0$ for any $a'_{<h} \neq a^t_{<h}$ and $a^t_h \neq a'_h$. If the last action a_t^H has incorrect answer, then

- For sparse reward setting, $\widehat{Q}_t([q, a_{<h}], a_h) = 0$ for any h and a_h.
- For our process supervision approach, the good prefix in this scheme are the action sequence that can lead to correct answer. Since we assume the correct sequence is unique on this tree, we assume this sequence as $[q, a_1^\star, \cdots, a_H^\star]$. In our reward scheme, given rollout $[q, a_1^t, \cdots, a_H^t]$, let $h_t^\star = \max_{h\in[H]}\left\{h : a^t_{h'} = a^\star_{h'}, \forall h' \leq h\right\}$ be the largest step number that the current rollout coincides with the correct sequence. From our reward scheme defined in Eq. (6) and set the good prefix as $[a_1^t, \cdots, a^t_{h_t^\star}]$, we have an additional reward α that is given to the end of $a^t_{h_t^\star}$. Thus, the Monte-Carlo estimated $\widehat{Q}_t([q, a_{<h}], a_h) = \alpha$ if $h \leq h_t^\star$ and $\widehat{Q}_t([q, a_{<h}], a_h) = 0$ when $h > h_t^\star$.

By induction, Eq. (9) has the following equivalent form

$$\pi_{t+1}(a_h|[q, a_{<h}]) \propto \pi_0(a_h|[q, a_{<h}]) \exp\left(\eta \sum_{k=1}^{t} \widehat{Q}_k([q, a_{<h}], a_h)\right) \tag{10}$$

where π_0 is the initial policy which is uniformly random as we assumed.

Lower Bound for Sparse Reward Setting. From the update rule above, the policy is only updated when the correct path is samlped and every action in this correct path is updated. Let N_t be the number of correct samples observed before round t, Eq. (10) only has update for the action in the correct path

$$\begin{aligned}\pi_{t+1}(a_h^\star|[q,a_{<h}^\star]) &= \frac{\exp\left(\eta\sum_{k=1}^t \widehat{Q}_k([q,a_{<h}^\star],a_h^\star)\right)}{\exp\left(\eta\sum_{k=1}^t \widehat{Q}_k([q,a_{<h}^\star],a_h^\star)\right)+\exp\left(\eta\sum_{k=1}^t \widehat{Q}_k([q,a_{<h}^\star],\bar{a}_h)\right)}\\ &= \frac{\exp\left(\eta\sum_{k=1}^t \widehat{Q}_k([q,a_{<h}^\star],a_h^\star)\right)}{\exp\left(\eta\sum_{k=1}^t \widehat{Q}_k([q,a_{<h}^\star],a_h^\star)\right)+\exp\left(\eta\sum_{k=1}^t \widehat{Q}_k([q,a_{<h}^\star],\bar{a}_h)\right)}\\ &= \frac{\exp\left(\eta N_t\right)}{\exp\left(\eta N_t\right)+1}\\ &= \frac{1}{1+\exp\left(-\eta N_t\right)}\end{aligned}$$

Since a correct sample can be observed if and only if a policy takes the right action at every layer, we have the probability of getting a correct sample from π_{t+1} is

$$\mathbb{P}_{o\sim\pi_{t+1}}\left(o \text{ is correct}\right) = \prod_{h=1}^H \pi_{t+1}(a_h^\star|[q,a_{<h}^\star]) = \left(\frac{1}{1+\exp\left(-\eta N_t\right)}\right)^H$$

To ensure $\mathbb{P}_{o\sim\pi_{t+1}}\left(o \text{ is correct}\right) \geq 1-\epsilon$, it is equivalent to ensure

$$\left(\frac{1}{1+\exp\left(-\eta N_t\right)}\right)^H \geq 1-\epsilon \Leftrightarrow N_t \geq \frac{1}{\eta}\ln\left(\frac{1}{\frac{1}{(1-\epsilon)^{\frac{1}{H}}}-1}\right)$$

Let $c(\epsilon) = \frac{1}{\eta}\ln\left(\frac{1}{\frac{1}{(1-\epsilon)^{\frac{1}{H}}}-1}\right)$, and τ_i be the round that the i-th correct sample is observed and $\tau_0 = 0$. To ensure $\mathbb{P}_{o\sim\pi_{t+1}}\left(o \text{ is correct}\right) \geq 1-\epsilon$, at least we need $\tau_{c(\epsilon)}$ samples. Let $W_i = \tau_{i+1}-\tau_i$, we have $\tau_c = \sum_{i=1}^{c-1} W_i = \tau_{i+1}-\tau_i$ for any $c \geq 1$. Since policy is not updated between τ_i and τ_{i+1}, $W_i \sim \text{Geom}\left(\pi_{\tau_i}\right)$ conditioned on history. Thus, for any $c \geq 1$, we have

$$\begin{aligned}\mathbb{E}\left[\tau_c\right] = \mathbb{E}\left[\sum_{i=0}^{c-1} W_i\right] = \sum_{i=0}^{c-1}\mathbb{E}\left[W_i\right] = \sum_{i=0}^{c-1}\frac{1}{\mathbb{P}_{o\sim\pi_{\pi_{\tau_i}}}\left(o \text{ is correct}\right)} &= \sum_{i=0}^{c-1}\left(1+\exp\left(-\eta i\right)\right)^H\\ &\geq 2^H\sum_{0=1}^{c-1}\exp\left(-\frac{\eta H i}{2}\right) \quad \text{(AM-GM)}\\ &= 2^H\frac{1-\exp\left(-\frac{\eta H c}{2}\right)}{1-\exp\left(-\frac{\eta H}{2}\right)}\\ &\geq \frac{2^{H+1}}{\eta H}\left(1-\exp\left(-\frac{\eta H c}{2}\right)\right)\\ &\qquad (1-e^{-x} \leq x \text{ for } x>0)\end{aligned}$$

Since

$$c(\epsilon) = \frac{1}{\eta}\ln\left(\frac{(1-\epsilon)^{\frac{1}{H}}}{1-(1-\epsilon)^{\frac{1}{H}}}\right) = \Theta\left(\frac{1}{\eta}\log\left(\frac{H}{\epsilon}\right)\right)$$

We have

$$\mathbb{E}\left[N^\star_{\text{sparse}}(\epsilon)\right] = \mathbb{E}\left[\tau_{c(\epsilon)}\right] \geq \frac{2^{H+1}}{\eta H}\left(1-\exp\left(-\frac{\eta H c(\epsilon)}{2}\right)\right) \geq \Omega\left(\frac{2^{H+1}}{\eta H}\left(1-\left(\frac{\epsilon}{H}\right)^{\frac{H}{2}}\right)\right) \geq \Omega\left(\frac{2^H}{\eta H}\right)$$
$$(\epsilon < \tfrac{1}{2})$$

Upper Bound with Process Supervision. Different from the sparse reward setting where the policy can only get update when the full correct path $[a_1^\star, a_2^\star, \cdots, a_H^\star]$ is sampled. In our scheme, a process reward model can identify the good prefix in a path that can lead to the correct answer and give this good prefix with α reward. Thus, even if the rollout is incorrect, the action in this path that matches the unique correct trajectory will be preserved, accelerating the identification of the correct path.

We will prove the sample complexity by induction for layers. The base case holds trivially because the learner starts from the root node with probability 1. For any round t, define $\mathcal{F}_t$ as the history of action and observation before t. Assume for any round $t \geq t_{h-1}$, we have $\pi_t([a_1^\star, \cdots, a_{h-1}^\star]|q) = \prod_{h'=1}^{h-1} \pi_t(a_{h'}^\star|[q, a_{<h'}^\star]) \geq 1 - \frac{h\epsilon}{H}$. Consider the reward scheme Eq. (6) and policy update rule Eq. (10), we have

$$\begin{aligned}\pi_{t+1}(a_h^\star|[q, a_{<h}^\star]) &= \frac{\exp\left(\eta \sum_{k=1}^t \widehat{Q}_k([q, a_{<h}^\star], a_h^\star)\right)}{\exp\left(\eta \sum_{k=1}^t \widehat{Q}_k([q, a_{<h}^\star], a_h^\star)\right) + \exp\left(\eta \sum_{k=1}^t \widehat{Q}_k([q, a_{<h}^\star], \bar{a}_h)\right)} \\ &= \frac{\exp\left(\eta \sum_{k=1}^t \widehat{Q}_k([q, a_{<h}^\star], a_h^\star)\right)}{\exp\left(\eta \sum_{k=1}^t \widehat{Q}_k([q, a_{<h}^\star], a_h^\star)\right) + \exp\left(\eta \sum_{k=1}^t \widehat{Q}_k([q, a_{<h}^\star], \bar{a}_h)\right)} \\ &= \frac{\exp\left(\eta\left(N_t + \alpha N_t^h\right)\right)}{\exp\left(\eta\left(N_t + \alpha N_t^h\right)\right) + 1} \\ &= \frac{1}{1 + \exp\left(-\eta\left(N_t + \alpha N_t^h\right)\right)}\end{aligned}$$

where we use N_t to denote the observed samples with correct answer till time t and N_t^h is the observed incorrect samples that contain $[a_1^\star, \cdots, a_{h-1}^\star]$ till time t. We have the following equivalent conditions

$$\pi_{t+1}(a_h^\star|[q, a_{<h}^\star]) \geq 1 - \frac{\epsilon}{H} \Leftrightarrow \frac{1}{1 + \exp\left(-\eta\left(N_t + \alpha N_t^h\right)\right)} \geq 1 - \frac{\epsilon}{H} \Leftrightarrow N_t + \alpha N_t^h \geq \frac{1}{\eta} \ln\left(\frac{1}{\frac{1}{(1-\frac{\epsilon}{H})^{\frac{1}{H}}} - 1}\right)$$

Let $c'(\epsilon) = \frac{1}{\eta\alpha} \ln\left(\frac{1}{\frac{1}{(1-\frac{\epsilon}{H})^{\frac{1}{H}}} - 1}\right)$, to ensure $\pi_{t+1}(a_h^\star|[q, a_{<h}^\star]) \geq 1 - \frac{\epsilon}{H}$, it is sufficient to have $N_t^h - N_{t_{h-1}}^h \geq c'(\epsilon)$. Define $t_h = \min\{t \mid N_t^h - N_{t_{h-1}}^h \geq c'(\epsilon)\}$. From the inductive hypothesis, we have that for any $t \geq t_{h-1}$, $\pi_t([a_1^\star, \cdots, a_{h-1}^\star]|q) \geq 1 - \frac{h\epsilon}{H}$, thus,

$$\mathbb{E}\left[t_h - t_{h-1} \mid \mathcal{F}_{t_{h-1}}\right] \leq \frac{c'(\epsilon)}{\frac{1}{2}\left(1 - \frac{h\epsilon}{H}\right)} \leq 4c'(\epsilon)$$

Therefore, for each h, given $\pi_{t_{h-1}}\left([a_1^\star, \ldots, a_{h-1}^\star] \mid q\right) \geq 1 - \frac{(h-1)\epsilon}{H}$, it suffices to collect at most $4c'$ additional samples in expectation to reach $\pi_{t_h}\left([a_1^\star, \ldots, a_h^\star] \mid q\right) \geq 1 - \frac{h\epsilon}{H}$. By induction over $h = 1, \ldots, H$, the total expected number of samples needed to achieve $\pi_t([a_1^\star, \ldots, a_H^\star] \mid q) \geq 1 - \epsilon$ is bounded by

$$t_H \leq 4Hc'(\epsilon) = \mathcal{O}\left(\frac{H}{\eta\alpha} \log \frac{H}{\epsilon}\right),$$

which yields the claimed upper bound $\mathbb{E}\left[N_{\text{dense}}^\star(\epsilon)\right] \leq \left(\frac{H}{\eta\alpha} \log \frac{H}{\epsilon}\right)$.

□

G The Choice of $\mathcal{RP}$ in Eq. (6)

In this section, we discuss the choice of reward prefix $\mathcal{RP}$. For completeness, we restate several notations and definition again. For a response o with token-level decomposition $(o_1, o_2, \dots, o_{|o|})$ and step-level decomposition $(s_1, s_2, \dots, s_{M_o})$, where each step s_m is itself a subsequence of tokens $s_m = (o_{t_m}, o_{t_m+1}, \dots, o_{t_{m+1}-1})$, $1 = t_1 < t_2 < \cdots < t_{M_o+1} = |o| + 1$, the process reward model operates at the step-level and identifies the first incorrect step. Let $M_{\text{err}} \in \{1, \dots, M_o\}$ denote the index of the first incorrect step, so that $s_1, \dots, s_{M_{\text{err}}-1}$ are labeled correct and $s_{M_{\text{err}}}$ is the first incorrect step. The *good prefix* $\mathcal{GP}(q, o) = (s_1, \dots, s_{M_{\text{err}}-1}) = (o_1, \dots, o_{t_{M_{\text{err}}}-1})$. We consider two approahces for $\mathcal{RP}$ below.

Simple Prefix: A natural definition of the reward prefix $\mathcal{RP}$ is every token before the first incorrect step. In this case, $\mathcal{RP}(q, o) = \mathcal{GP}(q, o)$. Using this simple-prefix selection, we plug $\mathcal{RP}(q, o)$ into our reward scheme in Eq. (6) with $\alpha = 0.5$ and the advantage normalization in Eq. (7) to finetune the Qwen3-4B-Base model. We report the resulting Average@16 performance (sampling 16 responses per question and averaging accuracy across benchmarks) in Table 6. Although the simple-prefix choice outperforms both GRPO and prior PRM-enhanced methods, the gain is limited.

Shorten Prefix: As discussed in Section 4, the simple prefix scheme can lead to reasoning step number inflation. To counteract it, we introduce a simple shorten-prefix strategy that trims a small suffix from the *good prefix* region for $\mathcal{RP}$. We choose a nonnegative cut length $c(q, o)$ and define $\mathcal{RP}_{\text{short}}(q, o) := (o_1, \dots, o_{t_{M_{\text{err}}}-c(q,o)-1})$ i.e., we discard the last $c(q, o)$ tokens from the simple prefix. If $c(q, o) > M_{\text{err}} - 1$, then $\mathcal{GP}_{\text{short}}(q, o) = \emptyset$. We refer to this strategy as shorten prefix. We test three different choice of $c(q, o)$ in Table 6. For **Fixed** method, we set $c(q, o) = 200$ for every question, indicating that we remove the last 200 tokens for every response in every question. For **Parial** method, we set $c(q, o) = 0.1|o|$, which means we remove the last $0.1|o|$ tokens for response o, and remain the first 90% tokens. For **Prompt** method, we set $c(q, o)$ to the the prompt length with question q, and remove the last $c(q, o)$ tokens from the simple prefix region. As shown in Figure 5, shorten prefix (we use the **Prompt** method for this figure) substantially reduces the number of steps in model outputs and effectively mitigates the reward-hacking behavior. Moreover, from Table 6, all these shorten prefix strategies outperform the simple prefix method, in which **Prompt** method has the best performance while **Pratial** method has least performance improvement. Thus, for all our method implementation except Table 6, we use shorten prefix (**Prompt** method).

G.1 Simple and Shorten Prefix Example

In this section, we present case studies comparing two prefix-rewarding schemes in Eq. (6): setting $\mathcal{GP}(q, o)$ as the whole *good prefix* (simple-prefix scheme) or setting $\mathcal{GP}(q, o)$ as the shorten *good prefix* (shorten-prefix scheme). More discussion on these two schemes can be found in Section 4 and Appendix G. When we apply the simple prefix scheme, the model tends to split each step into short sentences, causing the number of steps to grow rapidly. In contrast, when we apply the shortened prefix strategy without rewarding tail tokens from the *good prefix*, the number of steps remains stable and does not explode.

Question

Suppose that $f(x) = ax + b$ and $g(x) = -3x + 5$. If $h(x) = f(g(x))$ and $h^{-1}(x) = x + 7$, find $a - b$.

Simple Prefix

Step 1: We are given the functions $f(x) = ax + b$ and $g(x) = -3x + 5$.

Step 2: We need to find the composite function $h(x) = f(g(x))$.

Step 3: Substitute $g(x)$ into $f(x)$:

$$h(x) = f(g(x)) = f(-3x + 5)$$

Step 4: Since $f(x) = ax + b$, we have:

$$f(-3x + 5) = a(-3x + 5) + b$$

Step 5: Simplify the expression:

$$h(x) = -3ax + 5a + b$$

Step 6: We are given that $h^{-1}(x) = x + 7$.

Step 7: To find $h(x)$, we need to determine the inverse of $h(x) = -3ax + 5a + b$.

Step 8: Let $y = h(x) = -3ax + 5a + b$.

Table 6: **Average@16 of Different Choice of reward prefix $\mathcal{RP}$ in *Eq.* (6) and Baselines across benchmarks**. Simple Prefix method rewards all tokens before the first incorrect steps while Shorten Prefix method removes a small suffix from the simple-prefix region, and only reward this shorten part. Shorten Prefix method performs better.

Algorithm	**Benchmarks**								
	AIME-2025	AIME-2024	AMC-23	MATH-500	Minerva	Olympiadbench	Hmmt-feb-2025	Hmmt-feb-2024	**Average**
				Qwen3-4B-Base					
Base	5.2	7.7	35.9	58.7	21.6	28.7	1.3	2.3	20.2
GRPO	11.9	18.3	59.2	82.3	38.3	47.5	4.0	8.3	33.7
Mixed	12.3	14.9	53.1	80.4	38.5	45.4	3.5	6.0	31.8
RTS	14.0	17.5	54.1	80.6	36.9	46.4	1.3	8.0	32.4
Simple Prefix	16.5	17.5	59.8	81.4	37.9	47.3	**6.5**	8.1	34.4
Shorten Prefix (Fixed)	15.4	18.8	59.5	82.7	39.1	**48.1**	7.1	8.1	34.9
Shorten Prefix (Partial)	14.2	18.9	59.8	81.6	38.6	47.8	7.3	7.9	34.5
Shorten Prefix (Prompt)	**16.6**	**19.4**	**60.0**	**83.0**	**39.1**	47.7	6.0	**11.0**	**35.4**

Step 9: Solve for x in terms of y:

$$y = -3ax + 5a + b$$

Step 10: Isolate x:

$$y - 5a - b = -3ax$$

Step 11: Factor out x:

$$x = \frac{5a + b - y}{3a}$$

Step 12: Therefore, the inverse function $h^{-1}(y)$ is:

$$h^{-1}(y) = \frac{5a + b - y}{3a}$$

Step 13: We are given that $h^{-1}(x) = x + 7$.

Step 14: Equate the two expressions for $h^{-1}(x)$:

$$\frac{5a + b - x}{3a} = x + 7$$

Step 15: Multiply both sides by $3a$ to clear the denominator:

$$5a + b - x = 3a(x + 7)$$

Step 16: Expand the right-hand side:

$$5a + b - x = 3ax + 21a$$

Step 17: Collect all terms involving x on one side and constant terms on the other side:

$$5a + b - 21a = 3ax + x$$

Step 18: Simplify the equation:

$$-16a + b = x(3a + 1)$$

Step 19: For this equation to hold for all x, the coefficients of x and the constant terms must separately satisfy the equation.

Step 20: The coefficient of x on the right-hand side is $3a + 1$, so:

$$3a + 1 = 0$$

Step 21: Solve for a:

$$3a + 1 = 0$$

$$3a = -1$$

$$a = -\frac{1}{3}$$

Step 22: Substitute $a = -\frac{1}{3}$ back into the constant term equation:

$$-16a + b = 0$$

Step 23: Substitute $a = -\frac{1}{3}$:

$$-16\left(-\frac{1}{3}\right) + b = 0$$

$$\frac{16}{3} + b = 0$$

Step 24: Solve for b:

$$b = -\frac{16}{3}$$

Step 25: We need to find $a - b$:

$$a - b = -\frac{1}{3} - \left(-\frac{16}{3}\right)$$

$$a - b = -\frac{1}{3} + \frac{16}{3}$$

$$a - b = \frac{15}{3}$$

$$a - b = 5$$

Step 26: Output the final answer:

$$\boxed{5}$$

Shorten Prefix

To solve the problem, let's follow the steps systematically.
Step 1: Express $h(x)$ in terms of x.
Given:

$$f(x) = ax + b$$

$$g(x) = -3x + 5$$

$$h(x) = f(g(x))$$

Substitute $g(x)$ into $f(x)$:

$$h(x) = f(g(x)) = f(-3x + 5)$$

$$h(x) = a(-3x + 5) + b$$

$$h(x) = -3ax + 5a + b$$

So, $h(x)$ is a linear function:

$$h(x) = -3ax + (5a + b)$$

Step 2: Find the inverse function $h^{-1}(x)$.
Given $h^{-1}(x) = x + 7$, we can find $h(x)$ by switching x and $h^{-1}(x)$:

$$h(h^{-1}(x)) = x$$

$$h(x + 7) = x$$

Substitute $h(x) = -3ax + (5a + b)$ into the equation:

$$h(x + 7) = -3a(x + 7) + (5a + b)$$

$$= -3ax - 21a + 5a + b$$

$$= -3ax - 16a + b$$

Set this equal to x:

$$-3ax - 16a + b = x$$

Step 3: Equate the coefficients.
For the equation $-3ax - 16a + b = x$ to hold for all x, the coefficients of x and the constant terms must be equal:
1. Coefficient of x:

$$-3a = 1$$

$$a = -\frac{1}{3}$$

2. Constant term:

$$-16a + b = 0$$

$$-16\left(-\frac{1}{3}\right) + b = 0$$

$$\frac{16}{3} + b = 0$$

$$b = -\frac{16}{3}$$

Step 4: Compute $a - b$.

$$a - b = -\frac{1}{3} - \left(-\frac{16}{3}\right)$$

$$a - b = -\frac{1}{3} + \frac{16}{3}$$

$$a - b = \frac{15}{3}$$

$$a - b = 5$$

Final Answer:

$$\boxed{5}$$

H Hyperparameter Sensitivity and Ablation Study

The impact of different choices of α. In previous discussions, we fix $\alpha = 0.5$. To test the sensitivity of hyperparameter α, we choose $\alpha = 0.3, 0.5, 0.7, 0.9$ to test the performance difference. Table 7 and Table 8 show the Average@16 and Pass@K results for different α. We observe that the performance is stable when $\alpha = 0.3, 0.5, 0.7$ but get performance degeneration when α increases to 0.9. This implies that although the good prefix of incorrect steps worth preserving, weight them too large compared with the correct answer can hinder policy optimization.

Ablation on Standard Deviation and RELU Operator. We also do throughly experiments on different choice of standard deviation and RELU operator for advantage estimation.

Recall that for a group of responses $\{o^1, \cdots, o^G\}$, we define the average advantage of response o^i as

$$A_i(q) = \frac{1}{|o^i|}\sum_{i=1}^{|o^i|} A_{i,t}(q) = \begin{cases} 1 & o^i \text{ is correct} \\ \frac{\alpha|\mathcal{GP}(q, o^i)|}{|o^i|} & \text{otherwise} \end{cases}$$

Define $\mathrm{R}(x) = \mathrm{RELU}(x) = \max\{x, 0\}$. We consider the following four scenario for advantage estimation.

Table 7: **Average@16 for Different Choice of α**. Our Method Outperform all Baselines.

Algorithm	**Benchmarks**								**Average**
	AIME-2025	AIME-2024	AMC-23	MATH-500	Minerva	Olympiadbench	Hmmt-feb-2025	Hmmt-feb-2024	
Qwen3-4B-Base									
Base	5.2	7.7	35.9	58.7	21.6	28.7	1.3	2.3	20.2
GRPO	11.9	18.3	59.2	82.3	38.3	47.5	4.0	8.3	33.7
Ours ($\alpha = 0.3$)	**16.9**	18.5	**60.3**	82.3	38.9	47.8	6.0	11.5	35.3
Ours ($\alpha = 0.5$)	16.6	**19.4**	60.0	**83.0**	**39.1**	47.7	6.0	11.0	**35.4**
Ours ($\alpha = 0.7$)	15.0	18.9	60.0	82.8	38.6	**48.8**	**6.9**	**11.9**	35.4
Ours ($\alpha = 0.9$)	10.8	18.3	56.4	80.9	38.7	46.3	4.0	8.8	33.0

Table 8: **Different α for Average Pass@K Values on AIME 2025 and AIME 2024**

Algorithm	**Pass@K**							
	1	2	4	8	16	32	64	128
Qwen3-4B-Base								
Base	6.5	10.6	15.6	21.2	26.8	33.0	40.1	47.0
GRPO	15.4	20.3	24.6	28.8	33.3	38.3	44.0	49.9
Ours ($\alpha = 0.3$)	**18.4**	**23.2**	27.0	30.6	34.8	40.5	47.9	**55.4**
Ours ($\alpha = 0.5$)	17.4	23.0	**27.8**	**32.3**	**36.9**	**42.1**	**48.0**	53.4
Ours ($\alpha = 0.7$)	16.5	21.6	26.0	30.5	35.5	41.1	47.5	54.0
Ours ($\alpha = 0.9$)	13.9	18.1	22.9	28.2	34.0	40.3	47.2	54.8

w. std/w. RELU

$$\hat{A}_{i,t}(q)=\begin{cases} \mathrm{R}\left(\dfrac{A_{i,t}(q)-\mathrm{mean}(A_1(q),\ldots,A_G(q))}{\mathrm{std}\,(A_1(q),\ldots,A_G(q))}\right) \\ \qquad o^i \text{ is incorrect and } o_t^i \in \mathcal{GP}(q,o^i) \\ \dfrac{A_{i,t}(q)-\mathrm{mean}(A_1(q),\ldots,A_G(q))}{\mathrm{std}\,(A_1(q),\ldots,A_G(q))} \\ \qquad \text{otherwise} \end{cases}$$

w. std/w. o. RELU

$$\hat{A}_{i,t}(q) = \frac{A_{i,t}(q)-\mathrm{mean}(A_1(q),\ldots,A_G(q))}{\mathrm{std}\,(A_1(q),\ldots,A_G(q))}$$

w.o. std/w. RELU

$$\hat{A}_{i,t}(q)=\begin{cases} \mathrm{R}\,(A_{i,t}(q)-\mathrm{mean}(A_1(q),\ldots,A_G(q))) \\ \qquad o^i \text{ is incorrect and } o_t^i \in \mathcal{GP}(q,o^i) \\ A_{i,t}(q) - \mathrm{mean}(A_1(q),\ldots,A_G(q)) \\ \qquad \text{otherwise} \end{cases}$$

w.o. std/w.o. RELU

$$\hat{A}_{i,t}(q) = A_{i,t}(q) - \mathrm{mean}(A_1(q),\ldots,A_G(q))$$

The results are reported in Table 9. We additionally include GRPO without standard-deviation normalization as a baseline, which is also used in (Liu et al., 2025e). Our method uses the *shortened-prefix* (prompt-based) variant described in Appendix G. We observe that incorporating the standard deviation in advantage normalization can degrade performance. This is because our advantage estimates are non-uniform within a response, and the across-response standard deviation can become extremely small, yielding numerically inflated normalized advantages when dividing by it. We also find a capability-dependent effect of the ReLU operator: for lower-capability models (e.g., Qwen3-4B-Base and Qwen3-8B-Base), it is preferable to omit ReLU and allow negative gradients on the good prefix when a response group contains many correct samples; in contrast, for higher-capability models (e.g., Qwen3-4B), applying ReLU is beneficial as it prevents negative gradients on the good prefix.

I The use of AI

We use large language models to assist writing preprocessing code and polish writing.

Table 9: **Ablation on Standard Deviation and RELU Operator.** The highest number for each column is bold.

Algorithm	**Benchmarks**								
	AIME-2025	AIME-2024	AMC-23	MATH-500	Minerva	Olympiadbench	Hmmt-feb-2025	Hmmt-feb-2024	**Average**
Qwen3-4B-Base									
Base	5.2	7.7	35.9	58.7	21.6	28.7	1.3	2.3	20.2
GRPO	11.9	18.3	59.2	82.3	38.3	47.5	4.0	8.3	33.7
GRPO (w.o. std)	13.7	18.1	57.9	81.8	**39.4**	47.1	6.0	8.1	34.0
Ours (w. std/w. RELU)	15.8	17.7	56.1	81.6	38.1	46.1	5.0	10.4	33.9
Ours (w. std/w.o. RELU)	**17.3**	17.3	62.0	82.9	36.7	47.6	**6.5**	8.3	34.8
Ours (w.o. std/w. RELU)	15.8	17.3	**62.8**	82.2	39.1	**48.4**	5.8	6.3	34.7
Ours (w.o. std/w.o. RELU)	16.6	**19.4**	60.0	**83.0**	39.1	47.7	6.0	**11.0**	**35.4**
Qwen3-8B-Base									
Base	3.3	6.7	26.4	49.4	23.7	22.4	1.0	1.7	16.8
GRPO	17.2	22.3	63.4	85.2	41.5	51.5	5.4	6.4	36.6
GRPO (w.o. std)	19.2	21.7	62.3	85.2	**41.7**	51.3	6.7	7.5	37.0
Ours (w. std/w. RELU)	16.0	17.0	63.8	83.4	40.2	48.6	5.2	4.3	34.8
Ours (w. std/w.o. RELU)	15.8	20.2	**64.2**	84.5	40.2	50.0	5.6	6.7	35.9
Ours (w.o. std/w. RELU)	**19.6**	**22.7**	63.3	85.4	39.8	52.1	7.5	7.3	37.2
Ours (w.o. std/w.o. RELU)	18.3	**22.7**	63.3	**86.0**	40.4	**52.3**	**12.9**	**7.9**	**38.0**
Qwen3-4B (non-think)									
Base	18.1	19.5	66.7	82.5	38.1	51.3	10.0	10.6	37.1
GRPO	26.0	30.4	69.2	88.5	40.6	60.0	11.9	16.7	42.9
GRPO (w.o. std)	26.4	31.7	72.9	88.3	40.8	60.2	12.5	16.2	43.3
Ours (w. std/w. RELU)	24.8	28.1	72.8	88.3	**42.5**	58.9	13.1	17.2	43.2
Ours (w. std/w.o. RELU)	22.3	28.3	74.5	88.3	42.4	58.9	10.8	15.2	42.6
Ours (w.o. std/w. RELU)	**29.2**	**31.8**	**74.5**	88.0	41.7	**60.6**	12.7	**19.6**	**44.8**
Ours (w.o. std/w.o. RELU)	26.9	29.8	73.1	**88.7**	42.3	60.2	**14.4**	16.5	44.0