ON DESIGNING EFFECTIVE RL REWARD AT TRAINING TIME FOR LLM REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Reward models have been increasingly critical for improving the reasoning capability of LLMs. Existing research has shown that a well-trained reward model can substantially improve model performances *at inference time* via search or best-of-N votes. However, the potential of reward models during *RL training time* still remains largely under-explored. It is currently unclear whether these reward models can provide additional training signals to RL training that uses sparse success rewards, which verify the correctness of solutions. In this work, we evaluate popular reward models for RL training, including the Outcome-supervised Reward Model (ORM) and the Process-supervised Reward Model (PRM), and train a collection of LLMs for math problems using RL by combining these learned rewards with success rewards. Surprisingly, even though these learned reward models have strong inference-time performances, they may only bring marginal improvements or even hurt RL *training*, producing worse performances than LLMs trained with the success reward only. We find that *training collapse* easily occurs in RL training when PRM simply serves as reward shaping in addition to the success rewards. Our further analysis reveals two issues that may lead to the sub-optimal performance. Therefore, we introduce two novel reward refinement techniques, including the ***Clip*** and the ***Delta*** mechanisms, to tackle the identified issues. We evaluate our techniques with multiple reward models over a set of 1.5B and 7B LLMs on MATH and GSM8K benchmarks, where both ***Clip*** and ***Delta*** consistently enhance RL training. Finally, we also demonstrate that with a carefully designed reward function, pure RL training without any additional supervised tuning can further improve all the evaluated LLMs, including the state-of-the-art 7B LLM Qwen2.5-Math-7B-Instruct on MATH and GSM8K benchmarks.

1 INTRODUCTION

There is a recent trend to improve the reasoning ability of LLMs with learned reward models (Lightman et al., 2024; Wang et al., 2024b; Yu et al., 2024a; Zhang et al., 2024; Lee et al., 2024; Yang et al., 2024b; Luo et al., 2024; Chen et al., 2024c; Havrilla et al., 2024; Shao et al., 2024; Uesato et al., 2022). Recent research has been focusing on guiding search processes during inference (Lightman et al., 2024; Snell et al., 2024; Wang et al., 2024b), with two main categories of reward models: Outcome-supervised Reward Model (ORM) (Cobbe et al., 2021b; Yu et al., 2024a) and Process-supervised Reward Model (PRM) (Lightman et al., 2024; Wang et al., 2024b; Luo et al., 2024). ORM generates *outcome rewards* that estimate the *success rewards*, which evaluate the correctness of generated answers, enabling the selection of the most reliable answer from a pool of generated candidates. By contrast, PRM is trained to distinguish correct reasoning steps from incorrect ones and can provide step-level *process rewards* for search algorithms like Monte-Carlo Tree Search (Chen et al., 2024a) and beam search (Snell et al., 2024).

However, the potential of reward models in RL training for LLM reasoning is not yet fully explored. The most straightforward method for RL training in reasoning tasks is to optimize the success rewards. Some prior works further try the integration of a reward model into RL training (Havrilla et al., 2024; Wang et al., 2024b; Shao et al., 2024). Havrilla et al. (2024) finds that PPO training with a reward model only results in performance degeneration. In addition, some powerful LLMs that exhibit strong reasoning abilities such as the Qwen2.5-Math family (Yang et al., 2024b) and DeepseekMath-7B-RL (Shao et al., 2024) adopt RL training with reward models as a part of their

overall training process for mathematical reasoning. However, due to a lack of detailed analysis on the reward models, it remains unclear whether the reward models can provide additional training signals beyond what the success rewards offer for LLM reasoning.

In this work, we evaluate popular reward models, including ORM and PRM, as RL rewards on the challenging mathematical reasoning benchmark MATH (Hendrycks et al., 2021) and GSM8K (Cobbe et al., 2021a) by using PPO as the RL algorithm (Schulman et al., 2017). Surprisingly, we find that these reward models may only bring marginal benefits to RL training or even lead to performance degradation, yielding even worse results than LLMs trained with a sparse success reward only. We observe that outcome rewards consistently achieve similar training results as success rewards. We hypothesize that outcome rewards may not provide additional training signals since a more accurate success reward is accessible at *training* time. For PRM, we perform an in-depth analysis of the RL training process and observe that RL training easily collapses when simply using the PRM as reward shaping in addition to the success rewards. Through further case studies, we identify two possible causes, including the intrinsic biases of the PRM and a reward misspecification issue, which can be largely exploited by the training LLM to generate sub-optimal behavior patterns.

To tackle these challenges, we propose two novel techniques, i.e., ***Clip*** and ***Delta***, which refine the process rewards for effective RL training. In particular, the ***Clip*** mechanism mitigates the intrinsic biases of PRM. The ***Delta*** mechanism tackles the reward misspecification issue by optimizing single-step PRM rewards. Evaluation of these two techniques on synthetic reasoning trajectories demonstrates that they can effectively mitigate the tendency of training to favor the observed sub-optimal patterns. Finally, we conduct full RL training on a set of advanced 1.5B and 7B LLMs from the Qwen2 and Qwen2.5 families (Yang et al., 2024a;b) with different reward models. Our experiment results show that our proposed techniques effectively enhance RL training. Moreover, with a carefully crafted reward, RL training can improve all the evaluated LLMs, including the state-of-the-art 7B LLM Qwen2.5-Math-7B-Instruct on the challenging MATH and GSM8K (Hendrycks et al., 2021; Cobbe et al., 2021a) benchmarks.

2 RELATED WORK

Reinforcement Learning for LLMs. In RLHF, Reinforcement learning algorithms can effectively fine-tune LLMs to align with the preference of humans (Dong et al., 2023; Rafailov et al., 2024; Ouyang et al., 2022; Xu et al., 2024; Schulman et al., 2017), to improve the reasoning ability (Shao et al., 2024; Yang et al., 2024b) and coding skills (Wang et al., 2024a; Guo et al., 2024). PPO is the most widely used among the popular RL algorithms due to its robust performance across various domains (Ouyang et al., 2022; Xu et al., 2024). Xu et al. (2024) investigates the implementation details of PPO for dialogue tasks and coding tasks, revealing batch size as a critical factor for improving PPO performance in reinforcement learning from human feedback (RLHF). Our work addresses the challenge of designing RL rewards for LLM reasoning.

Reward Learning for LLMs. Learned reward models are widely adopted in RLHF to align LLMs with human preferences (Dong et al., 2023; Rafailov et al., 2024; Ouyang et al., 2022). In RLHF, reward models are trained on binary preference datasets collected from human annotators, following the Bradley-Terry model (Bradley & Terry, 1952). In reasoning tasks involving reliable solution checkers, two main approaches are the Outcome-supervised Reward Model (ORM) (Cobbe et al., 2021b; Yu et al., 2024a) and the Process-supervised Reward Model (PRM) (Lightman et al., 2024; Wang et al., 2024b; Luo et al., 2024). An ORM predicts the correctness of the final answer. A PRM estimates whether the steps so far are correct. Despite the successful applications of reward models, *reward hacking* is a broadly observed issue in learned reward models (Skalse et al., 2022; Singhal et al., 2023; Casper et al., 2023). Through RL training, the LLM may learn to generate high-reward outputs that could not fulfill the intended objectives. Several approaches have been proposed to tackle the reward hacking issue, including disentangling the length aspect of reward modeling (Chen et al., 2024b; Shen et al., 2023), reward ensemble (Eisenstein et al., 2024; Rame et al., 2024), length penalty (Singhal et al., 2023), length normalization (Meng et al., 2024), and various PPO implementation tricks (Singhal et al., 2023; Zheng et al., 2023). In this work, we investigate how to effectively use PRM/ORM as rewards in RL training for LLM reasoning, and

our proposed techniques are related to reward shaping methods in standard RL. (Ng et al., 1999; Harutyunyan et al., 2019; Arjona-Medina et al., 2019; Patil et al., 2020; Widrich et al., 2021) .

Improving Reasoning Ability of LLMs. To improve the reasoning ability of LLMs, prior works have focused on several different aspects, including pre-training (Yang et al., 2024b; Achiam et al., 2023; Anil et al., 2023), prompting (Han et al., 2024; Yuan et al., 2024; Wu et al., 2024), search during inference-time (Lightman et al., 2024; Wang et al., 2024b; Yu et al., 2024a; Zhang et al., 2024; Yang et al., 2024b; Luo et al., 2024; Chen et al., 2024c), and fine-tuning (Wang et al., 2024b; Shao et al., 2024; Yang et al., 2024b; Shah et al., 2024; Tang et al., 2024; Yu et al., 2024b). Pre-training methods focus on enriching the data distribution to cover a large amount of rationals and pre-training the LLM over the dataset. The prompting methods elicit the reasoning ability of LLMs through dedicated prompting strategies and automatic agent frameworks. Inference-time search utilizes learned reward models to guide the selection of promising solutions. PRM and ORM could be combined with different search strategies such as Best-of-N, Monte-Carlo Tree Search (Chen et al., 2024a), and Beam Search (Snell et al., 2024). Finally, fine-tuning methods include training the LLM on high-quality question-answer data (Yu et al., 2024b; Shah et al., 2024; Yue et al., 2024) and optimizing the reasoning ability with reinforcement learning (Yang et al., 2024b; Shao et al., 2024; Wang et al., 2024b). In this work, we study how to effectively combine dense and sparse rewards in RL training for reasoning tasks.

3 PRELIMINARY

Language Model. An LLM is represented as a policy $\pi_\theta(s|q)$ parameterized by θ. In reasoning tasks, π_θ generates a solution s given a question q. In addition to the question, q usually also contains a prompt to elicit chain-of-thought reasoning. The solution s is structured with a list of reasoning steps and thus can be viewed from two perspectives, including tokens and steps. From the perspective of tokens, s consists of T tokens, $s = (s_1, s_2, \cdots, s_T)$. From the perspective of steps, s consists of K reasoning steps, $s = (s^{(1)}, s^{(2)}, \cdots, s^{(K)})$ where $s^{(k)}$ denotes the k-th reasoning step. For convenience, we use $p^{(k)} = (s^{(1)}, s^{(2)}, \cdots, s^{(k)})$ to denote the solution prefix up to the k-th step. In practice, reasoning steps can be parsed with rule-based detectors, enforcing strict output formats, or special tokens (Chen et al., 2024a; Wang et al., 2024b; Lightman et al., 2024).

Reward Modeling. In RLHF, the reward models are usually trained with binary preferences (Bradley & Terry, 1952). In reasoning tasks where the correctness of solutions is accessible, reward models can be trained under the supervision of such ground-truth correctness. In reasoning tasks, two primary methods for reward modeling are the Process-supervised Reward Model (PRM) and the Outcome-supervised Reward Model(ORM).

Given a question q and a prefix $s_{1:t}$, an ORM estimates the likelihood the prefix would lead to a correct answer. A standard approach to train an ORM is by first sampling solutions for questions from a dataset with an LLM and then labeling the correctness of each solution. The ORM r_{outcome} is then trained with the following objective,

$$\mathcal{L}_{\text{ORM}} = \mathbb{E}_{q,s\sim\mathcal{D}}\left[\sum_{t=1}^{T} \text{Loss}(\text{Correct}(q,s), r_{\text{outcome}}(q, s_{1:t}))\right]$$

where $\text{Correct}(q, s)$ is a binary value indicating the correctness of solution s, t enumerates each token of the solution s, and Loss denotes the loss function. In practice, the loss function could be binary cross-entropy loss or square-error loss, and we can choose to train ORM on the full sequence or only the last token.

In contrast, Process-supervised Reward Model (PRM) estimates the correctness of individual reasoning steps. PRM is trained with the following objective,

$$\mathcal{L}_{\text{PRM}} = \mathbb{E}_{q,p^{(k)},y_k\sim\mathcal{D}}\left[\text{Loss}(y_k, r_{\text{process}}(q, p^{(k)}))\right]$$

where y_k is the label for the partial solution $p^{(k)}$ and Loss is the loss function. In practice, binary cross entropy loss is usually adopted. Prior works have investigated several ways to annotate the process labels, including human annotators (Lightman et al., 2024) and automatic annotation with LLMs (Wang et al., 2024b; Luo et al., 2024).

Reinforcement Learning for LLM Reasoning. We assume access to the correctness of a solution during training. We use $\text{Correct}(q, s)$ to indicate the correctness of solution s to question q, which is also referred to as the *success reward* for RL training. An LLM can be fine-tuned to optimize the success reward by using Reinforcement Learning with Kullback-Leibler divergence,

$$J_r(\pi_\theta) = \mathbb{E}_{q\sim\mathcal{D}, s\sim\pi_\theta}\left[\text{Correct}(q, s) - \beta \log \frac{\pi_\theta(s|q)}{\pi_{ref}(s|q)}\right] \tag{1}$$

where π_{ref} is the reference model for regularizing π_θ. Optimizing the success reward only provides a sparse training signal because the reward is provided at the end of the sequence. Alternatively, we can also combine dense rewards with the success reward. The RL objective with dense rewards becomes,

$$J_r(\pi_\theta) = \mathbb{E}_{q\sim\mathcal{D}, s\sim\pi_\theta}\left[\alpha \cdot \sum_{t=1}^{|s|} r(q, s_{1:t}) + \text{Correct}(q, s) - \beta \log \frac{\pi_\theta(s|q)}{\pi_{ref}(s|q)}\right] \tag{2}$$

where r denotes the dense reward and α is a coefficient for the dense reward. For example, a PRM r_{process} can provide dense feedback at the end of reasoning steps, formally represented as $r(q, p^{(k)}) = r_{\text{process}}(q, p^{(k)})$ for any partial solution $p^{(k)}$. In the subsequent sections, we would refer to the rewards generated by ORM as *outcome rewards*, and the rewards generated by PRM as *PRM rewards* to avoid ambiguity between *process rewards* and *dense rewards*.

4 RL Reward for LLM Reasoning

In this section, we conduct a systematic study on reward design to aid LLM in learning better reasoning skills through RL training. We follow the RL objective with dense rewards in Eq. (2) and specifically focus on the effective design of dense rewards. As discussed in Sec. 3, the ground-truth correctness, $\text{Correct}(p, s)$, serves to provide the sparse rewards, and the dense rewards could be provided by a reward model.

4.1 Evaluating RL Training with Learned Reward Models

We first consider two straightforward approaches to apply ORM and PRM to provide rewards in addition to success rewards for RL training. Formally, we consider the following rewards,

- **Solution-Level Outcome Reward (OR):** In the RL training process of Yang et al. (2024b), an ORM provides an estimation of correctness as reward shaping. Note that this is not the case for dense rewards since ORM only produces rewards at the end of the sequence. For a question q and a solution s,

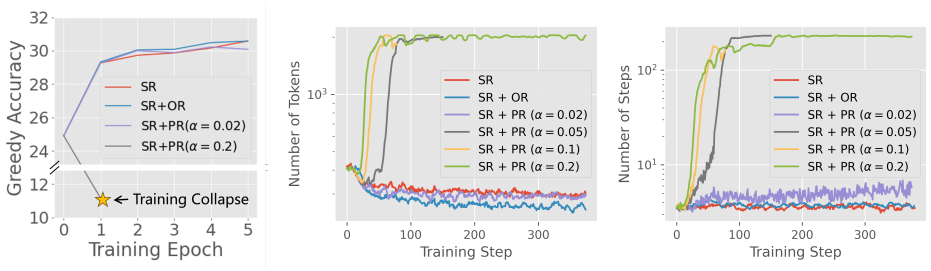
$$r(q, s) = r_{\text{outcome}}(q, s) \tag{3}$$

- **Step-Level Process Reward (PR):** A PRM can provide step-level feedback for RL training. For any solution prefix $p^{(k)}$, dense rewards are the rewards outputted by a PRM,

$$r(q, p^{(k)}) = r_{\text{process}}(q, p^{(k)}) \tag{4}$$

Experiment Setup. We carry out our study on the challenging mathematical reasoning benchmark, MATH (Hendrycks et al., 2021). We use PPO as the RL algorithm and Qwen2-1.5B-Instruct (Yang et al., 2024a) as the base model. For ORM, we sample solutions with the base model and train ORM with binary cross-entropy loss. For PRM, we follow Wang et al. (2024b) to generate process labels with automatic annotation[1]. The ORM and PRM both use Qwen2-1.5B-Instruct as the base model.

[1]Implementation details can be found in Sec. 5



(a) Greedy Accuracy (b) Avg. Number of Tokens (c) Avg. Number of Steps

Figure 1: (a) Test greedy accuracy during RL training with a combination of success reward and OR/PR on Qwen2-1.5B-Instruct. SR denotes the success reward. α denotes the reward shaping coefficient. None of OR/PR can surpass training with success rewards. (b) Generation length during RL training. (c) Step count during RL training.

Results. Surprisingly, we find these reward functions may not benefit RL training, yielding even worse inference-time performances than LLMs trained with a sparse success reward only, as shown in Fig. 1(a). To further investigate the cause of performance degradation, Fig. 1(b) reports the change in the generation length and the number of reasoning steps during training. Although introducing OR improves the sample efficiency, SR+OR shows a similar evaluation accuracy to adopting a sparse success reward only (SR). We hypothesize this is because an outcome reward may not be able to provide additional information beyond the success reward during training time. On the other hand, when using PRM for RL training, training easily collapses when α is large and we observe a significant increment in the generation length and the number of reasoning steps during RL training. When α is small, SR+PR only achieves sub-optimal performance compared with SR.

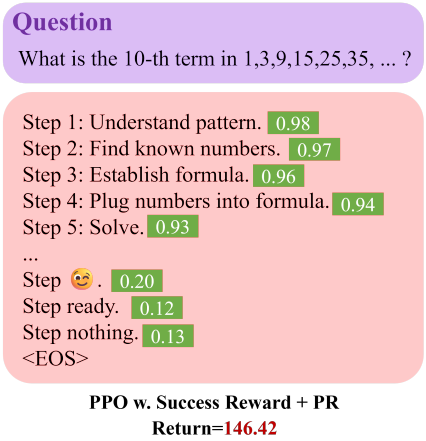


Figure 2: Case study of PR. PRM provides rewards at the end of each step. For PR with a large α, the LLM learns to generate many reasoning steps that do not contribute to problem-solving to achieve a high return through RL training.

Case Study for PR. For PR with a large α ($\alpha \geq 0.05$), a case study of the generated samples reveals that the LLM learns to obtain high rewards with some specific patterns without faithfully optimizing the ground-truth correctness through RL training. In the generated solutions of PR, there are many short reasoning steps, but these steps only contain unnecessary or meaningless information that does not contribute to problem-solving. As the generation length increases, the model outputs only a single word or even emoji.

Analysis for PR. The rewards of unnecessary reasoning steps are positive and could even be large, as shown in the case study (Fig. 2). The LLM learns to exploit this phenomenon by generating more reasoning steps, resulting in a higher return. We further confirm the behavior through some synthetic reasoning trajectories (Fig. 4(a) and Fig. 4(b)), where PR demonstrates extremely larger returns. We identify two key observations through further case studies,

Observation 1: PRM has *intrinsic biases* to output positive and even high rewards to sub-optimal reasoning steps. In the left part of Fig. 3, we illustrate a case study of biases of PRM. The PRM could assign high values to steps with simple patterns, such as unnecessary and repetitive steps, even achieving higher rewards than the optimal correct steps. On the other hand, incorrect and nonsense steps could also receive small positive rewards. Consequently RL training would encourage the LLM to generate more sub-optimal steps.

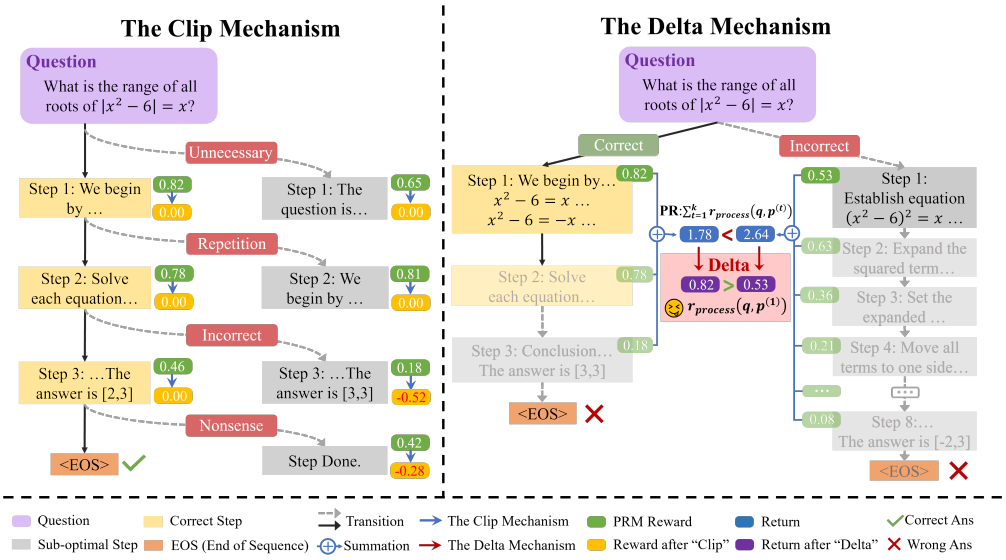


Figure 3: **Left: A case study of the intrinsic biases of PRM & the effect of the Clip mechanism.** PRM may assign high values to steps with simple patterns, such as unnecessary steps and repetitive ones, even higher than the correct steps. For incorrect and nonsense steps, PRM may assign small positive values. *The Clip mechanism can mitigate the intrinsic biases of PRM, preventing the LLM from obtaining high rewards through these undesired patterns and penalizing steps that have low PRM rewards.* **Right: A reward misspecification case of PR showing that RL training promotes an incorrect step & the effect of the Delta mechanism.** In this case, the left solution starts with two correct steps but has an incorrect answer. The right solution starts with an incorrect step and also has an incorrect answer. Although the first correct step in the left solution has a higher PRM reward than the first incorrect step in the right solution, RL training with PR would prefer the incorrect step. This is because the incorrect step receives a higher return than the correct step since the right solution accumulates PRM rewards of much more steps. *The Delta mechanism ensures the steps promoted by RL training are aligned with the PRM, which promotes the correct step in this case.*

Observation 2: RL training may mistakenly prefer an incorrect step with a low PRM reward, as shown by the case study in the right part of Fig. 3. Although the PRM successfully assigns a low PRM reward to the incorrect reasoning step, RL training would mistakenly encourage the incorrect step. We refer to this issue as the *reward misspecification* (Pan et al., 2022) issue since the RL objective of PR does not align with the desired target, i.e. better reasoning skills, even when the PRM can accurately assign higher rewards to better reasoning steps. Furthermore, We find such an issue also exists in PRMs that are trained with human-labeled data. (See Appendix F)

Here are two key takeaways regarding applying ORM and PRM in RL training,

Takeway for ORM. For ORM, it only marginally improves the sample efficiency of RL training compared with the sparse success reward. We hypothesize this is because, when a success reward is available during training time, ORM does not provide additional supervision signal and should not be a preferred choice at RL training time. We also remark that ORM does not suffer from training collapse since OR only uses ORM to generate sparse rewards, and the ORM outputs a 0-1 value, naturally ensuring an upper-bounded objective.

Takeway for PRM. Simply adopting PRM rewards as dense rewards would easily lead to training collapse during RL training. Although PRM provides useful training signals for intermediate steps, issues including the intrinsic biases of the learned PRM and the reward misspecification issue could cause the LLM to generate sub-optimal behavior patterns through RL training.

4.2 Constructing Effective RL Rewards with PRM

Since ORM does not provide dense feedback for RL training and may lack additional information beyond the success reward during training, PRM can be a more suitable source for dense rewards.

However, as analyzed in Sec. 4.1, the intrinsic biases of PRM and the issue of reward misspecification pose challenges for adopting PRM to promote better reasoning skills in RL training. To effectively unleash the potential of PRM in RL training, we introduce two novel techniques designed to utilize PRM in RL training effectively,

The *Clip* mechanism. We propose the *Clip* mechanism to mitigate the intrinsic biases of PRM. To prevent the LLM from exploiting the reward model to achieve high rewards with undesired patterns such as repetition and unnecessary reasoning, a straightforward idea is to upper-bound high rewards by a selected threshold η. For steps with low PRM rewards, we penalize such steps with a negative reward $r_{\text{process}}(q, p^{(k)}) - \eta$. Formally, with a threshold η,

$$r(q, p^{(k)}) = \min(r_{\text{process}}(q, p^{(k)}) - \eta, 0) \tag{5}$$

If a suitable η is chosen, the majority of the reasoning steps would receive a reward of 0, and only steps with low r_{process} would have a negative reward. In practice, for each question in a training batch, by default we set η to be the average PRM rewards of all reasoning steps.

The *Delta* mechanism. We also introduce the *Delta* mechanism to tackle reward misspecification issue. The *Delta* mechanism subtracts the rewards between adjacent steps. Specially, the reward for the last reasoning step is dropped since the success reward would be sufficient to provide guidance for the last reasoning step. Formally, for a solution prefix $p^{(k)}$,

$$r(q, p^{(k)}) = \begin{cases} r_{\text{process}}(q, p^{(k)}) - r_{\text{process}}(q, p^{(k+1)}) & \text{if } k < K-1 \\ r_{\text{process}}(q, p^{(k)}) & \text{if } k = K-1 \\ 0 & \text{if } k = K \end{cases} \tag{6}$$

A nice property of the Delta mechanism is that it ensures the return starting from any intermediate solution step $p^{(k)}$ is $\alpha \cdot r_{\text{process}}(q, p^{(k)}) + \text{Correct}(q, s)$, which is unaffected by the PRM rewards of future steps. Following (Sutton, 2018), the policy gradient of combining the PRM rewards processed with the Delta mechanism and sparse success rewards is (informally) given by,

$$\nabla_\theta J_r(\pi_\theta) = \mathbb{E}_{q\sim\mathcal{D}, s\sim\pi_\theta(\cdot|q)}[\nabla_\theta \log \pi_\theta(s|q) \cdot \text{Correct}(q, s) \tag{7}$$

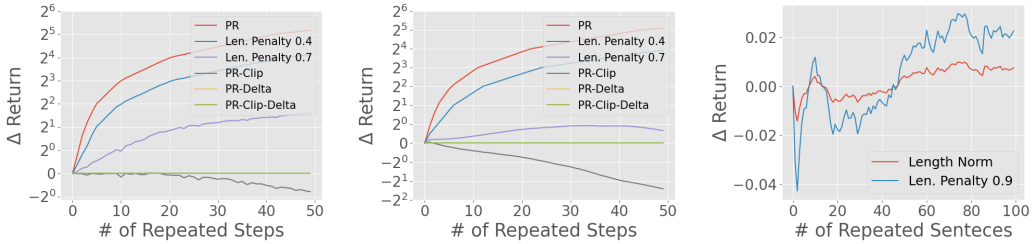
$$+ \alpha \cdot \underbrace{\sum_{k=1}^{K-1} \nabla_\theta \log \pi_\theta(s^{(k)}|q, p^{(k-1)}) \cdot r_{process}(q, p^{(k)})}_{\text{Effect of the Delta mechanism}}] + \text{KL term} \tag{8}$$

Consequently, RL training would focus on optimizing single-step PRM rewards. The theoretical analysis can be found in Appendix. E.

These mechanisms can be used individually or in combination. In practice, we consider three approaches incorporating these mechanisms: (1) *PR-Clip*, which applies the Clip mechanism on the PRM rewards, (2) *PR-Delta*, which applies the Delta mechanism, and (3) *PR-Clip-Delta*, which first applies the Clip mechanism and then the Delta mechanism.

We further perform evaluation on synthetic solutions that exhibit repetitive patterns in different ways. As shown in Fig. 4(b) and Fig. 4(a), the Clip mechanism and the Delta mechanism can both successfully limit the upper bound of the returns on these synthetic solutions. Additionally, the Clip mechanism imposes increasingly smaller returns as the length of the repetitive pattern grows.

Other Practices. We also compare with some adopted practices to avoid reward hacking in prior works (Singhal et al., 2023), including length normalization and length penalty. More details can be found in Appendix C. Length normalization normalizes the rewards for each solution. Length penalty imposes a constant penalty for each step. As illustrated in Fig. 4, imposing length penalty and length normalization could still favor the undesired repetition modes over correct solutions. We also investigate standard normalization for PRM as employed by Shao et al. (2024), which we find would lead to training instability. More details can be found in Sec. 5.2.



(a) Repeat nonsense steps (b) Repeat intermediate steps (c) Repeat last sentences

Figure 4: Difference between the returns of synthetic solutions and the ground-truth solution. The synthetic solutions are constructed from the ground-truth solution by (a) repeatedly adding nonsense steps to the end of the sequence, (b) repeating an intermediate step, and (c) repeating the last sentence in the solution. A positive return difference indicates the repetitive patterns are favored over the ground-truth solution. Both PR and length penalty can gain significantly high returns in (a) and (b). In (c), when sufficient repetitions are inserted, length normalization and length penalty would assign a higher return to the synthetic solution than to the ground-truth solution.

5 Experiments

5.1 Experiment Setup

Training Dataset. We conduct RL training on the MathInstruct (Yue et al., 2024) dataset. In particular, we only use the questions and the golden answers in the dataset while the provided solutions are not used for training. We use Qwen2-7B-Instruct to sample 16 answers for each question in the training dataset and keep those questions that have both correct and wrong answers. To train an ORM, binary cross entropy loss is adopted. For PRM training, we follow Wang et al. (2024b) to generate automatic process annotations by using Qwen2-7B-Instruct as the completer. Specifically, for each step in the generated samples, we sample 8 solutions starting from the solution prefix. This step is labeled as correct if any of these 8 solutions is correct.

Benchmarks & Metrics. We carry out our evaluation on the GSM8K (Cobbe et al., 2021a) and MATH (Hendrycks et al., 2021) datasets. For evaluation metrics, we report the *Greedy* and *Sampling* scores, which correspond to adopting greedy decoding and sampling with temperature of 1 as generation strategies, respectively.

Base Models. Our experiments are taken over a series of large language models from the Qwen2 (Yang et al., 2024a) family and the state-of-the-art LLMs for mathematical reasoning, Qwen2.5 (Yang et al., 2024b) family. Specifically, we use various 1.5B and 7B LLMs, including general and math-specific models. For general models, we consider Qwen2-1.5B-Instruct and Qwen2-7B-Instruct. For math-specific models, we consider Qwen2-Math-1.5B-Instruct, Qwen2.5-Math-1.5B-Instruct, Qwen2-Math-7B-Instruct and Qwen2.5-Math-7B-Instruct. Note that these LLMs already equip sufficient instruction following ability and we do not perform any further supervised fine-tuning. Lastly, the PRM is trained with the same base model as the actor model.

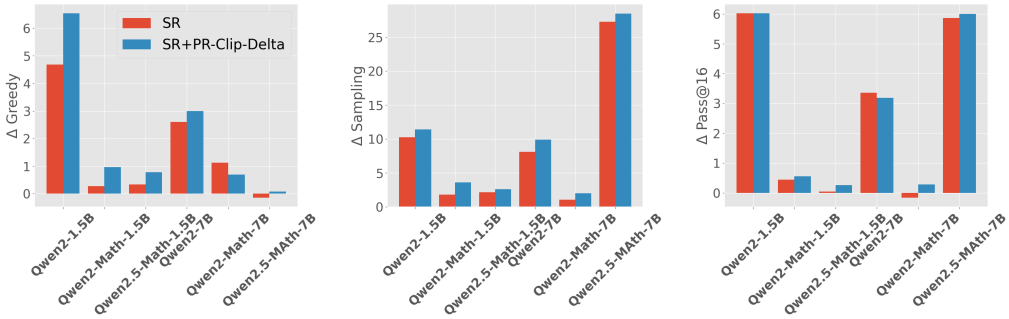
RL Training We adopt the Proximal Policy Optimization (PPO) implementation of ReaLHF (Mei et al., 2024), which supports LLM fine-tuning with dense rewards. The detailed hyperparameters and training setup are listed in Appendix. D.

5.2 Ablation Study

The Clip & Delta Mechanisms Our ablation study of the Clip and Delta mechanisms is presented in Table 1. We also consider a standard normalization variant of PR (Shao et al., 2024), denoted as PR-Normed. PPO training with OR can not surpass training with a sparse success reward. PR only achieves sub-optimal performance. Similarly, the performance of PR-Normed also decreases in the latter epochs. Consequently, none of OR, PR, and PR-Normed can achieve higher greedy decoding accuracy than training with a success reward. On the other hand, the Delta mechanism successfully stabilizes RL training, surpassing training with a success reward. Finally, by combining

Method	Greedy	Sampling
Qwen2-1.5B-Instruct	24.90	16.79
Success Reward	30.58	27.05
SR + OR	30.57	27.12
SR + PR (**E4**)	30.22	27.46
SR + PR-Normed (**E2**)	29.66	27.14
SR + PR-Normed (**E5**)	12.36	12.84
SR + PR-Clip	30.30	**28.40**
SR + PR-Delta	30.68	27.96
SR + PR-Clip-Delta	**31.44**	28.20

Table 1: Ablation study of various reward functions with Qwen2-1.5B-Instruct. **E2** denotes the results of the 2-nd epoch. Unless otherwise specified, we report the accuracy of final checkpoint.



(a) Δ Greedy (b) Δ Sampling (c) Δ Pass@16

Figure 5: Performance improvement of PPO training over the base LLMs using success rewards and further using PR-Clip-Delta as dense rewards. All LLMs are the instruction following model, and the "-Instruct" suffices are omitted for improved clarity. Adding PR-Clip-Delta as dense rewards consistently improves RL training with sparse success rewards only.

the Clip mechanism and the Delta mechanism, PR-Clip-Delta demonstrates the best greedy decoding accuracy. Training curves of all approaches are provided in Appendix. A.2

Effect of PR-Clip-Delta We compare the performance improvements of PPO training over the base LLMs when using a success reward and additionally using PR-Clip-Delta as dense rewards in Fig. 5. In addition to Greedy and Sampling scores, we also consider the Pass@16 score, which we believe can roughly estimate the upper bound of the model's capacity. Using PR-Clip-Delta as dense rewards can consistently improve RL training, across all LLMs and all evaluation metrics, except the greedy decoding accuracy on Qwen2-Math-7B-Instruct. This suggests that applying the Clip mechanism and the Delta mechanism can effectively utilize the PRM to guide the LLM in learning better reasoning skills during RL training. We report the detailed numbers in Appendix A.

5.3 Main Results

Main Results Our main results are summarized in Table. 2. RL training consistently improves the performance of the base model across all the models we test, even on the state-of-the-art 1.5B model, Qwen2.5-Math-1.5B-Instruct, and 7B model, Qwen2.5-Math-7B-Instruct. For 1.5B models, Qwen2-1.5B-Instruct obtains the most significant performance improvement. Through RL training with PR-Clip-Deta as reward function, the best 1.5B model, Qwen2.5-Math-1.5B-Instruct achieves 87.34% and 76.78% greedy decoding accuracy on GSM8K and MATH benchmark respectively, indicating 2.20% and 0.78% improvement of accuracy over the base model. For 7B models, building on the strongest 7B LLM, Qwen2.5-Math-7B-Instruct, RL training with dense reward further boosts the performance and achieves 95.6% and 83.38% greedy decoding accuracy on GSM8K and MATH benchmarks, respectively, surpassing several baselines. It is noteworthy that Qwen2.5-Math-7B-Instruct is already trained using RL, highlighting the effectiveness of PR-Clip-Delta.

Performance Improvement The performance improvement of RL training varies across models with different amounts of parameters and different strengths. In general, weaker models gain higher

Model	GSM8K		MATH	
	Greedy	Sampling	Greedy	Sampling
GPT-4o-2024-08-06	92.9	-	81.1	-
DeepSeekMath-7B-RL	88.2	-	52.4	-
Internlm2-math-plus-7B	84.0	-	54.4	-
Mathstral-7B-v0.1	84.9	-	56.6	-
NuminaMath-7B-CoT	75.4	-	55.2	-
Llama-3.1-8B-Instruct	76.6	-	47.2	-
1.5B Models				
Qwen2-1.5B-Instruct	50.19	44.58	24.90	16.79
+ PPO w. SR	67.70$^{\uparrow 17.51}$	65.50$^{\uparrow 20.92}$	30.58$^{\uparrow 4.68}$	27.05$^{\uparrow 10.26}$
+ PPO w. (SR + PR-Clip-Delta)	**68.76**$^{\uparrow 18.57}$	**66.19**$^{\uparrow 21.61}$	**31.44**$^{\uparrow 6.54}$	**28.20**$^{\uparrow 11.41}$
Qwen2-Math-1.5B-Instruct	83.62	81.50	69.98	64.51
+ PPO w. SR	84.61$^{\uparrow 0.99}$	83.93$^{\uparrow 2.43}$	70.26$^{\uparrow 0.28}$	66.29$^{\uparrow 1.78}$
+ PPO w. (SR + PR-Clip-Delta)	**85.67**$^{\uparrow 2.05}$	**84.76**$^{\uparrow 3.26}$	**70.94**$^{\uparrow 0.96}$	**68.13**$^{\uparrow 3.62}$
Qwen2.5-Math-1.5B-Instruct	85.14	82.11	76.00	72.05
+ PPO w. SR	86.73$^{\uparrow 1.59}$	85.82$^{\uparrow 3.71}$	76.34$^{\uparrow 0.34}$	74.22$^{\uparrow 2.17}$
+ PPO w. (SR + PR-Clip-Delta)	**87.34**$^{\uparrow 2.20}$	**85.97**$^{\uparrow 3.86}$	**76.78**$^{\uparrow 0.78}$	**74.63**$^{\uparrow 2.58}$
7B Models				
Qwen2-7B-Instruct	86.88	80.44	57.54	48.27
+ PPO w. SR	87.72$^{\uparrow 0.84}$	86.81$^{\uparrow 6.37}$	60.14$^{\uparrow 2.60}$	56.39$^{\uparrow 8.12}$
+ PPO w. (SR + PR-Clip-Delta)	**87.64**$^{\uparrow 0.76}$	**87.34**$^{\uparrow 6.90}$	**60.54**$^{\uparrow 3.00}$	**58.17**$^{\uparrow 9.90}$
Qwen2-Math-7B-Instruct	89.61	89.23	75.30	72.09
+ PPO w. SR	89.46$^{\downarrow 0.15}$	90.07$^{\uparrow 0.84}$	**76.42**$^{\uparrow 1.12}$	73.12$^{\uparrow 1.03}$
+ PPO w. (SR + PR-Clip-Delta)	**90.90**$^{\uparrow 1.29}$	**90.14**$^{\uparrow 0.91}$	76.00$^{\uparrow 0.70}$	**74.09**$^{\uparrow 2.00}$
Qwen2.5-Math-7B-Instruct	95.60	80.74	83.30	52.76 [2]
+ PPO w. SR	95.45$^{\downarrow 0.15}$	**95.07**$^{\uparrow 14.33}$	83.16$^{\downarrow 0.14}$	79.95$^{\uparrow 27.19}$
+ PPO w. (SR + PR-Clip-Delta)	**95.60**$^{0.00}$	**95.07** $^{\uparrow 14.33}$	**83.38**$^{\uparrow 0.08}$	**81.22**$^{\uparrow 28.46}$

Table 2: Greedy and Sampling scores on GSM8K and MATH benchmarks. PPO training using sparse success rewards and PR-Clip-Delta as dense rewards consistently improve all evaluated LLMs, including the state-of-the-art 7B LLMs, Qwen2.5-Math-7B-Instruct. For sampling decoding, we adopt the temperature of 1.0.

performance improvements than stronger models. Comparing the improvements of Greedy and Sampling scores, the improvements of Sampling score are larger than those of Greedy score across all LLMs, resulting in a smaller gap between Sampling and Greedy scores.

6 Conclusion

In this work, we investigate designing dense rewards with a process-supervised reward model in RL training to improve the reasoning ability of LLMs. We examine some popular reward models and identify the issue of reward hacking, which manifests as the generation of nonsensical texts or unnecessary reasoning steps. The reward hacking issue can be mitigated with our proposed techniques, using the Clip mechanism to prevent the LLM from exploiting the reward model and the Delta mechanism to ensure a bounded RL objective. We show that the proposed techniques can be utilized to apply Process-supervised Reward Models for improved RL training.

Limitations. Limited by computation resources, our experiments are conducted over 1.5B&7B LLMs, while evaluations on larger LLMs could further help verify our proposed techniques. Also, it is an interesting direction to perform various inference-time search strategies with the LLMs trained with PPO, which could help further understand whether RL training can improve search performance. Furthermore, we believe that with the support of more powerful reward models, RL training can bring greater benefits to LLM reasoning.

[2]For sampling accuracy, we find that Qwen-2.5-math-Instruct is likely to generate strange characters, leading to poor sampling accuracy.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.

Jose A Arjona-Medina, Michael Gillhofer, Michael Widrich, Thomas Unterthiner, Johannes Brandstetter, and Sepp Hochreiter. Rudder: Return decomposition for delayed rewards. *Advances in Neural Information Processing Systems*, 32, 2019.

Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

Stephen Casper, Xander Davies, Claudia Shi, Thomas Krendl Gilbert, Jérémy Scheurer, Javier Rando, Rachel Freedman, Tomasz Korbak, David Lindner, Pedro Freire, Tony Tong Wang, Samuel Marks, Charbel-Raphael Segerie, Micah Carroll, Andi Peng, Phillip Christoffersen, Mehul Damani, Stewart Slocum, Usman Anwar, Anand Siththaranjan, Max Nadeau, Eric J Michaud, Jacob Pfau, Dmitrii Krasheninnikov, Xin Chen, Lauro Langosco, Peter Hase, Erdem Biyik, Anca Dragan, David Krueger, Dorsa Sadigh, and Dylan Hadfield-Menell. Open problems and fundamental limitations of reinforcement learning from human feedback. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=bx24KpJ4Eb. Survey Certification.

Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. Alphamath almost zero: process supervision without process. *arXiv preprint arXiv:2405.03553*, 2024a.

Lichang Chen, Chen Zhu, Jiuhai Chen, Davit Soselia, Tianyi Zhou, Tom Goldstein, Heng Huang, Mohammad Shoeybi, and Bryan Catanzaro. ODIN: Disentangled reward mitigates hacking in RLHF. In *Forty-first International Conference on Machine Learning*, 2024b. URL https://openreview.net/forum?id=zcIV8OQFVF.

Zhipeng Chen, Kun Zhou, Wayne Xin Zhao, Junchen Wan, Fuzheng Zhang, Di Zhang, and Ji-Rong Wen. Improving large language models via fine-grained reinforcement learning with minimum editing constraint. *arXiv preprint arXiv:2401.06081*, 2024c.

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

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

Hanze Dong, Wei Xiong, Deepanshu Goyal, Yihan Zhang, Winnie Chow, Rui Pan, Shizhe Diao, Jipeng Zhang, KaShun SHUM, and Tong Zhang. RAFT: Reward ranked finetuning for generative foundation model alignment. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=m7p5O7zblY.

Jacob Eisenstein, Chirag Nagpal, Alekh Agarwal, Ahmad Beirami, Alexander Nicholas D'Amour, Krishnamurthy Dj Dvijotham, Adam Fisch, Katherine A Heller, Stephen Robert Pfohl, Deepak Ramachandran, Peter Shaw, and Jonathan Berant. Helping or herding? reward model ensembles mitigate but do not eliminate reward hacking. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=5u1GpUkKtG.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

Vernon Toh Yan Han, Ratish Puduppully, and Nancy F. Chen. Veritymath: Advancing mathematical reasoning by self-verification through unit consistency. In *AI for Math Workshop @ ICML 2024*, 2024. URL https://openreview.net/forum?id=S9utaRXaZt.

Anna Harutyunyan, Will Dabney, Thomas Mesnard, Mohammad Gheshlaghi Azar, Bilal Piot, Nicolas Heess, Hado P van Hasselt, Gregory Wayne, Satinder Singh, Doina Precup, et al. Hindsight credit assignment. *Advances in neural information processing systems*, 32, 2019.

Alexander Havrilla, Yuqing Du, Sharath Chandra Raparthy, Christoforos Nalmpantis, Jane Dwivedi-Yu, Eric Hambro, Sainbayar Sukhbaatar, and Roberta Raileanu. Teaching large language models to reason with reinforcement learning. In *AI for Math Workshop @ ICML 2024*, 2024. URL https://openreview.net/forum?id=mjqoceuMnI.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL https://openreview.net/forum?id=7Bywt2mQsCe.

Jung Hyun Lee, June Yong Yang, Byeongho Heo, Dongyoon Han, and Kang Min Yoo. Token-supervised value models for enhancing mathematical reasoning capabilities of large language models. *arXiv preprint arXiv:2407.12863*, 2024.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

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

Zhiyu Mei, Wei Fu, Kaiwei Li, Guangju Wang, Huanchen Zhang, and Yi Wu. Realhf: Optimized rlhf training for large language models through parameter reallocation. *arXiv preprint arXiv:2406.14088*, 2024.

Yu Meng, Mengzhou Xia, and Danqi Chen. Simpo: Simple preference optimization with a reference-free reward. *arXiv preprint arXiv:2405.14734*, 2024.

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

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

Alexander Pan, Kush Bhatia, and Jacob Steinhardt. The effects of reward misspecification: Mapping and mitigating misaligned models. *arXiv preprint arXiv:2201.03544*, 2022.

Vihang P Patil, Markus Hofmarcher, Marius-Constantin Dinu, Matthias Dorfer, Patrick M Blies, Johannes Brandstetter, Jose A Arjona-Medina, and Sepp Hochreiter. Align-rudder: Learning from few demonstrations by reward redistribution. *arXiv preprint arXiv:2009.14108*, 2020.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36, 2024.

Alexandre Rame, Nino Vieillard, Leonard Hussenot, Robert Dadashi, Geoffrey Cideron, Olivier Bachem, and Johan Ferret. WARM: On the benefits of weight averaged reward models. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=s7RDnNUJy6.

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

Vedant Shah, Dingli Yu, Kaifeng Lyu, Simon Park, Nan Rosemary Ke, Michael Mozer, Yoshua Bengio, Sanjeev Arora, and Anirudh Goyal. Ai-assisted generation of difficult math questions. *arXiv preprint arXiv:2407.21009*, 2024.

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

Wei Shen, Rui Zheng, Wenyu Zhan, Jun Zhao, Shihan Dou, Tao Gui, Qi Zhang, and Xuanjing Huang. Loose lips sink ships: Mitigating length bias in reinforcement learning from human feedback. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://openreview.net/forum?id=qq6ctdUwCX.

Prasann Singhal, Tanya Goyal, Jiacheng Xu, and Greg Durrett. A long way to go: Investigating length correlations in rlhf. *arXiv preprint arXiv:2310.03716*, 2023.

Joar Skalse, Nikolaus Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward gaming. *Advances in Neural Information Processing Systems*, 35:9460–9471, 2022.

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

Richard S Sutton. Reinforcement learning: An introduction. *A Bradford Book*, 2018.

Zhengyang Tang, Xingxing Zhang, Benyou Wang, and Furu Wei. Mathscale: Scaling instruction tuning for mathematical reasoning. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=Kjww7ZN47M.

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

Ke Wang, Houxing Ren, Aojun Zhou, Zimu Lu, Sichun Luo, Weikang Shi, Renrui Zhang, Linqi Song, Mingjie Zhan, and Hongsheng Li. Mathcoder: Seamless code integration in LLMs for enhanced mathematical reasoning. In *The Twelfth International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=z8TW0ttBPp.

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

Michael Widrich, Markus Hofmarcher, Vihang Prakash Patil, Angela Bitto-Nemling, and Sepp Hochreiter. Modern hopfield networks for return decomposition for delayed rewards. In *Deep RL Workshop NeurIPS 2021*, 2021.

Zhenyu Wu, Qingkai Zeng, Zhihan Zhang, Zhaoxuan Tan, Chao Shen, and Meng Jiang. Large language models can self-correct with minimal effort. In *AI for Math Workshop @ ICML 2024*, 2024. URL https://openreview.net/forum?id=mmZLMs4l3d.

Shusheng Xu, Wei Fu, Jiaxuan Gao, Wenjie Ye, Weilin Liu, Zhiyu Mei, Guangju Wang, Chao Yu, and Yi Wu. Is DPO superior to PPO for LLM alignment? a comprehensive study. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=6XH8R7YrSk.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024a.

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

Fei Yu, Anningzhe Gao, and Benyou Wang. Ovm, outcome-supervised value models for planning in mathematical reasoning. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 858–875, 2024a.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng YU, Zhengying Liu, Yu Zhang, James Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. In *The Twelfth International Conference on Learning Representations*, 2024b. URL https://openreview.net/forum?id=N8N0hgNDRt.

Lifan Yuan, Ganqu Cui, Hanbin Wang, Ning Ding, Xingyao Wang, Jia Deng, Boji Shan, Huimin Chen, Ruobing Xie, Yankai Lin, Zhenghao Liu, Bowen Zhou, Hao Peng, Zhiyuan Liu, and Maosong Sun. Advancing LLM reasoning generalists with preference trees. In *AI for Math Workshop @ ICML 2024*, 2024. URL https://openreview.net/forum?id=2Y1iiCqM5y.

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. MAmmoTH: Building math generalist models through hybrid instruction tuning. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=yLClGs770I.

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. *arXiv preprint arXiv:2408.15240*, 2024.

Rui Zheng, Shihan Dou, Songyang Gao, Yuan Hua, Wei Shen, Binghai Wang, Yan Liu, Senjie Jin, Qin Liu, Yuhao Zhou, et al. Secrets of rlhf in large language models part i: Ppo. *arXiv preprint arXiv:2307.04964*, 2023.

A ADDITIONAL RESULTS

A.1 MAIN RESULTS

In Tab. 3 and Tab. 4, we report the results of RL training on different base models, including those with success rewards and after applying PR-Clip-Delta.

Model	Method	Math		
		Greedy	Sample	Pass@16
Qwen2-1.5B-Instruct	Basemodel	24.90	16.79	55.68
	Success Reward	30.58$^{\uparrow 4.68}$	27.05$^{\uparrow 10.26}$	61.70$^{\uparrow 6.02}$
	+ PR-Clip-Delta	31.44$^{\uparrow 6.54}$	28.20$^{\uparrow 11.41}$	61.70$^{\uparrow 6.02}$
Qwen2-Math-1.5B-Instruct	Basemodel	69.98	64.51	88.02
	Success Reward	70.26$^{\uparrow 0.28}$	66.29$^{\uparrow 1.78}$	88.46$^{\uparrow 0.44}$
	+ PR-Clip-Delta	70.94$^{\uparrow 0.96}$	68.13$^{\uparrow 3.62}$	88.58$^{\uparrow 0.56}$
Qwen2.5-Math-1.5B-Instruct	Basemodel	76.00	72.05	90.50
	Success Reward	76.34$^{\uparrow 0.34}$	74.22$^{\uparrow 2.17}$	90.54$^{\uparrow 0.04}$
	+ PR-Clip-Delta	76.78$^{\uparrow 0.78}$	74.63$^{\uparrow 2.58}$	90.76$^{\uparrow 0.26}$
Qwen2-7B-Instruct	Basemodel	57.54	48.27	80.04
	Success Reward	60.14$^{\uparrow 2.60}$	56.39$^{\uparrow 8.12}$	83.40$^{\uparrow 3.36}$
	+ PR-Clip-Delta	60.54$^{\uparrow 3.00}$	58.17$^{\uparrow 9.90}$	83.22$^{\uparrow 3.18}$
Qwen2-Math-7B-Instruct	Basemodel	75.30	72.09	91.24
	Success Reward	76.42$^{\uparrow 1.12}$	73.12$^{\uparrow 1.03}$	91.08$^{\downarrow 0.16}$
	+ PR-Clip-Delta	76.00$^{\uparrow 0.70}$	74.09$^{\uparrow 2.00}$	91.52$^{\uparrow 0.28}$
Qwen2.5-Math-7B-Instruct	Basemodel	83.3	52.76	86.6
	Success Reward	83.16$^{\downarrow 0.14}$	79.95$^{\uparrow 27.19}$	92.46$^{\uparrow 5.86}$
	+ PR-Clip-Delta	83.38$^{\uparrow 0.08}$	81.22$^{\uparrow 28.46}$	92.60$^{\uparrow 6.00}$

Table 3: Results on MATH test set

In Fig. 6, we report the greedy accuracy on MATH test set of different training epochs, where epoch-0 means the base model (i.e., Qwen2-1.5B-Instruct). The introduction of PR-norm caused the model's accuracy to drop significantly starting from the third epoch.

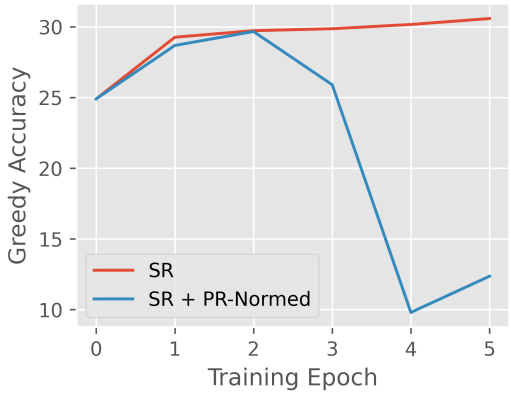


Figure 6: Greedy accuracy on MATH test set during the training process.

Model	Method	GSM8K	
		Greedy	Sample
Qwen2-1.5B-Instruct	Basemodel	50.19	44.58
	Success Reward	67.70$^{\uparrow 17.51}$	65.50$^{\uparrow 20.92}$
	+ PR-Clip-Delta	68.76$^{\uparrow 18.57}$	66.19$^{\uparrow 21.61}$
Qwen2-Math-1.5B-Instruct	Basemodel	83.62	81.50
	Success Reward	84.61$^{\uparrow 0.99}$	83.93$^{\uparrow 2.43}$
	+ PR-Clip-Delta	85.67$^{\uparrow 2.05}$	84.76$^{\uparrow 3.26}$
Qwen2.5-Math-1.5B-Instruct	Basemodel	85.14	82.11
	Success Reward	86.73$^{\uparrow 1.59}$	85.82$^{\uparrow 3.71}$
	+ PR-Clip-Delta	87.34$^{\uparrow 2.20}$	85.97$^{\uparrow 3.86}$
Qwen2-7B-Instruct	Basemodel	86.88	80.44
	Success Reward	87.72$^{\uparrow 0.84}$	86.81$^{\uparrow 6.37}$
	+ PR-Clip-Delta	87.64$^{\uparrow 0.76}$	87.34$^{\uparrow 6.90}$
Qwen2-Math-7B-Instruct	Basemodel	89.61	89.23
	Success Reward	89.46$^{\downarrow 0.15}$	90.07$^{\uparrow 0.84}$
	+ PR-Clip-Delta	90.90$^{\uparrow 1.29}$	90.14$^{\uparrow 0.91}$
Qwen2.5-Math-7B-Instruct	Basemodel	95.60	80.74
	Success Reward	95.45$^{\downarrow 0.15}$	95.07$^{\uparrow 14.33}$
	+ PR-Clip-Delta	95.60$^{\uparrow 0.00}$	95.07$^{\uparrow 14.33}$

Table 4: Results on GSM8K test set

A.2 Training Curves

We list training curves of all methods on Qwen2-1.5B-Instruct here.

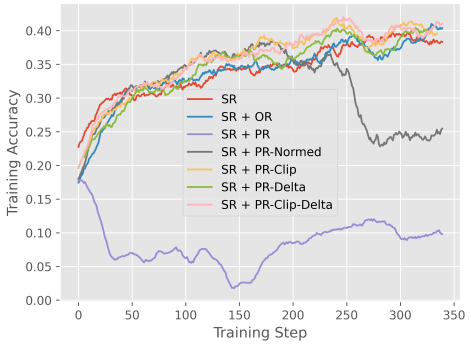


Figure 7: Training accuracy of all baselines and the proposed methods on Qwen2-1.5B-Instruct

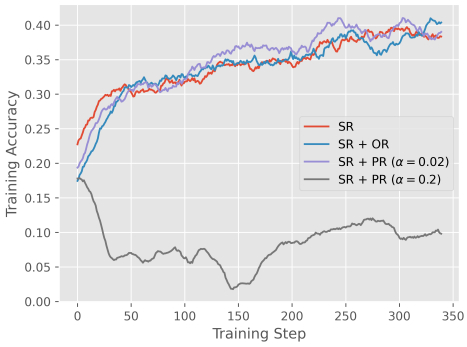


Figure 8: Training accuracy of SR, SR+OR, and SR+PR on Qwen2-1.5B-Instruct

Figure 9: Training accuracy of SR and SR+PR-Normed on Qwen2-1.5B-Instruct

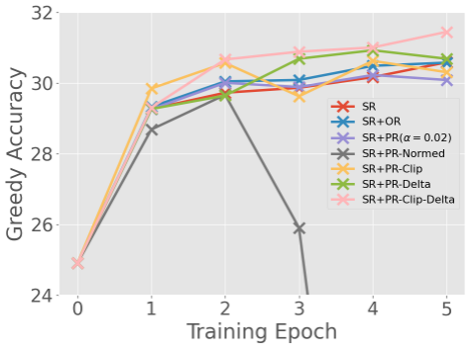


Figure 10: Test accuracy of all methods across training epochs on Qwen2-1.5B-Instruct

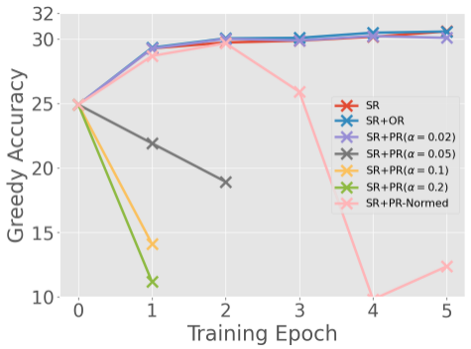


Figure 11: Test accuracy of all baselines across training epochs on Qwen2-1.5B-Instruct

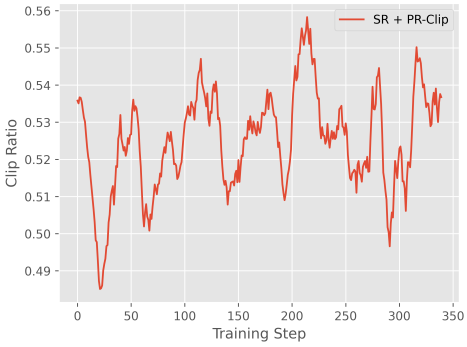


Figure 12: Clip ratio of PR-Clip during RL training. PR-Clip indeed provides dense rewards for RL training.

A.3 Ablation Studies

We perform ablation study for the reward shaping coefficient α in Tab. 5. The ablation study for threshold η in PR-Clip is provided in Fig. 13.

	Epoch 1	Epoch 2	Epoch 3	Epoch 4	Epoch 5
SR	29.26	29.72	29.86	30.16	30.58
SR+PR ($\alpha = 0.02$)	29.25	30.00	29.88	30.22	30.08
SR+PR ($\alpha = 0.05$)	21.90	18.92	/	/	/
SR+PR ($\alpha = 0.1$)	14.10	/	/	/	/
SR+PR ($\alpha = 0.2$)	11.16	/	/	/	/

Table 5: Test accuracy of SR vs. SR+PR with different reward shaping coefficient α across training epochs

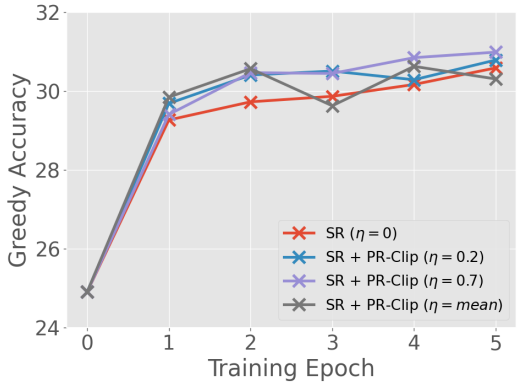


Figure 13: Ablation study of the threshold η of PR-Clip on Qwen2-1.5B-Instruct. Greedy accuracy is reported. 'η =mean' means that we set η to be the average PRM rewards of all reasoning steps related to a question in a training batch. A constant η offers more stable improvement.

B SYNTHETIC EXAMPLE

In Fig. 4, we insert nonsense steps or repeat the last sentences in the following manner.

Prompt:

Tom found a box of 220 baseball cards. Being a thoughtful older brother, he wanted to save at least 50 percent of the cards for himself and split the remaining cards evenly among his 5 younger brothers. What is the lowest possible number of cards that Tom can save for himself that will also allow each of his 5 younger brothers to get an equal number of cards? Please reason step by step, and put your final answer within \boxed{}.

Answer by Qwen2-1.5B-instruct:

Step 1: 50% of 220 cards

$$50\% \times 220 = 0.50 \times 220 = 110$$

Therefore, Tom would save 110 of the cards for himself.

Step 2: Remaining cards that Tom would like to distribute evenly among his 5 younger brothers

$$220 - 110 = 110$$

Tom would have 110 cards left to distribute evenly among his 5 younger brothers.

Step 3: Forming sets of cards and evenly distributing

If the 110 remaining cards are to be divided into 5 sets of equal cards, each set must contain:

$$\frac{110}{5} = 22$$

Therefore, each of Tom's 5 younger brothers would get 22 cards, but since each set also must have 22 cards, the lowest possible number of cards for Tom to save so that each brother gets an equal number of cards means Tom would save:

$$5 \times 22 = 110$$

The lowest possible number of cards that Tom can save for himself that will allow each of his 5 younger brothers to get an equal number of cards is 110. This way, Tom will save enough cards to permit a remainder that will also allow one card to be shared by each of his brothers out of the total of 110 cards.

The answer is: $\boxed{110}$.

Repeat Nonsense

Step 1: 50% of 220 cards

$$50\% \times 220 = 0.50 \times 220 = 110$$

...

Step 3: Forming sets of cards and evenly distributing ...

Step done.

Step done.

Step end.

Step end.

Step show.

...

Repeat Mid Step

Step 1: 50% of 220 cards

$$50\% \times 220 = 0.50 \times 220 = 110$$

...

Step 2: Remaining cards that Tom would like to distribute evenly among his 5 younger...

Step 2: Remaining cards that Tom would like to distribute evenly among his 5 younger...

Step 2: Remaining cards that Tom would like to distribute evenly among his 5 younger...

...

Step 3: Forming sets of cards and evenly distributing ...

The answer is: $\boxed{110}$.

Repeat Last Sentence

Step 1: 50% of 220 cards

$$50\% \times 220 = 0.50 \times 220 = 110$$

...

Step 3: Forming sets of cards and evenly distributing ...

The answer is: $\boxed{110}$.

The answer is: $\boxed{110}$.

The answer is: $\boxed{110}$.

The answer is: $\boxed{110}$.

...

C BASELINES

Length Normalization. Length normalization normalizes the rewards for each solution. Formally,

$$r(q, p^{(k)}) = \frac{1}{K} r_{\text{process}}(q, p^{(k)})$$

Length Penalty. Length penalty imposes a constant penalty for each step.

$$r(q, p^{(k)}) = r_{\text{process}}(q, p^{(k)}) - k * c_{\text{penalty}}$$

D HYPER-PARAMETERS AND TRAINING SETUP

PPO Hyperparameters and Training Setup. Following prior practices (Shao et al., 2024; Xu et al., 2024), we adopt a large batch size and sample multiple solutions for each question within a batch. We find a large batch size and multiple sampling critical to the overall performance of RL training. For 1.5B models, there are 1024 questions, and 8 solutions are sampled for each question in a batch, leading to a batch size of 1024×8. For 7B models, the batch size is 4096×8. Each training batch is split into 4 minibatches. We apply a KL penalty coefficient of 0.1, a coefficient of 1 for dense rewards, and a coefficient of 5 for successful rewards. For the reward threshold η in the Clip mechanism, by default the average value of PRM rewards of all reasoning steps related to one question in a training batch. The learning rates of 1B and 7B actor models are 1e-6 and 1e-5, respectively, while all critic models use a learning rate of 5e-6. We use Adam optimizer weight decay of 0.05. The 1.5B models are trained on a cluster of 4 machines, each with 8 Nvidia H100 GPUs, for approximately 8 hours. The 7B models are trained on a cluster of 8 machines, each with 8 Nvidia H100 GPUs, for approximately 20 hours.

Implementation of Clip and Delta. Both the Clip and the Delta mechanisms are straightforward to integrate into existing workflows. For the Clip mechanism, its implementation involves computing the mean of the reward as a threshold after reward calculation, followed by applying the formula specified in Eq. 5. This additional step is computationally lightweight and seamlessly fits within the existing reward processing pipeline. The Delta mechanism requires computing the difference between rewards from two adjacent steps, a process that is both conceptually simple and computationally efficient. As such, neither method introduces significant overhead, ensuring their ease of adoption.

E THEORETICAL ANALYSIS

E.1 RL TRAINING WITH SR+PR-DELTA

Lemma 1 (Policy gradient of RL training with SR+PR-Delta) *Following (Sutton, 2018), the policy gradient of the RL objective combining the PRM rewards processed with the Delta mech-*

anism and the sparse success reward (SR+PR-Delta) is given by,

$$\nabla_\theta J_r(\pi_\theta) = \mathbb{E}_{q\sim\mathcal{D},s\sim\pi_\theta(\cdot|q)}[\sum_{k=1}^{K}\nabla_\theta \log \pi_\theta(s^{(k)}|q,p^{(k-1)}) \cdot (\alpha \cdot r_{process}(q,p^{(k)}) \cdot \mathbb{I}[k<K] \tag{9}$$

$$+ \mathit{Correct}(q,s) - \beta \cdot \sum_{k'=k}^{K} \log \frac{\pi_\theta(s^{(k')}|q,p^{(k'-1)})}{\pi_{ref}(s^{(k')}|q,p^{(k'-1)})})] \tag{10}$$

Proof 1 *Following (Sutton, 2018), the vanilla policy gradient is given by,*

$$\nabla_\theta J_r(\pi_\theta) = \mathbb{E}_{q\sim\mathcal{D},s\sim\pi_\theta(\cdot|q)}[\sum_{k=1}^{K}\nabla_\theta \log \pi_\theta(s^{(k)}|q,p^{(k-1)})G_k] \tag{11}$$

where G_k is the return from the k-th reasoning step.
By the formula of the Delta mechanism (Eq. 6), we have,

$$\begin{aligned}
G_k &= \mathit{Correct}(q,s) + \alpha \cdot \sum_{k'=k}^{K} r(q,p^{(k')}) - \beta \cdot \sum_{k'=k}^{K} \log \frac{\pi_\theta(s^{(k')}|q,p^{(k'-1)})}{\pi_{ref}(s^{(k')}|q,p^{(k'-1)})}) \\
&= \mathit{Correct}(q,s) + \alpha \cdot \sum_{k'=k}^{K-1} (r_{process}(q,p^{(k')}) - r_{process}(q,p^{(k'+1)} \cdot \mathbb{I}[k+1<K]) \\
&\quad - \beta \cdot \sum_{k'=k}^{K} \log \frac{\pi_\theta(s^{(k')}|q,p^{(k'-1)})}{\pi_{ref}(s^{(k')}|q,p^{(k'-1)})}) \\
&= \mathit{Correct}(q,s) + \alpha \cdot \left(\sum_{k'=k}^{K-1} (r_{process}(q,p^{(k')}) - \sum_{k'=k+1}^{K-1} (r_{process}(q,p^{(k')})\right) \\
&\quad - \beta \cdot \sum_{k'=k}^{K} \log \frac{\pi_\theta(s^{(k')}|q,p^{(k'-1)})}{\pi_{ref}(s^{(k')}|q,p^{(k'-1)})}) \\
&= \mathit{Correct}(q,s) + \alpha \cdot r_{process}(q,p^{(k)}) \cdot \mathbb{I}[k<K] - \beta \cdot \sum_{k'=k}^{K} \log \frac{\pi_\theta(s^{(k')}|q,p^{(k'-1)})}{\pi_{ref}(s^{(k')}|q,p^{(k'-1)})})
\end{aligned}$$

Therefore,

$$\nabla_\theta J_r(\pi_\theta) = \mathbb{E}_{q\sim\mathcal{D},s\sim\pi_\theta(\cdot|q)}[\sum_{k=1}^{K}\nabla_\theta \log \pi_\theta(s^{(k)}|q,p^{(k-1)}) \cdot (\alpha \cdot r_{process}(q,p^{(k)}) \cdot \mathbb{I}[k<K] \tag{12}$$

$$+ \mathit{Correct}(q,s) - \beta \cdot \sum_{k'=k}^{K} \log \frac{\pi_\theta(s^{(k')}|q,p^{(k'-1)})}{\pi_{ref}(s^{(k')}|q,p^{(k'-1)})})] \tag{13}$$

Lemma 2 (Policy gradient of RL training with SR) *Following (Sutton, 2018), the vanilla policy gradient of using the sparse success reward only is given by,*

$$\nabla_\theta J_r(\pi_\theta) = \mathbb{E}_{q\sim\mathcal{D},s\sim\pi_\theta(\cdot|q)}[\sum_{k=1}^{K}\nabla_\theta \log \pi_\theta(s^{(k)}|q,p^{(k-1)}) \cdot (\mathit{Correct}(q,s) \tag{14}$$

$$- \beta \cdot \sum_{k'=k}^{K} \log \frac{\pi_\theta(s^{(k')}|q,p^{(k'-1)})}{\pi_{ref}(s^{(k')}|q,p^{(k'-1)})})] \tag{15}$$

Proof 2 *Omitted.*

E.2 PRM AS VALUE

In this section, we are going to show the connection between PRM training and Q learning. We will show that PRMs can be interpreted as Q functions or value functions.

Notation. To train a PRM, the following elements are required,

- **A PRM** $r_{process}(q, p)$ where q is the question and p is a partial solution prefix.
- **A prompt dataset** of pairs of questions and partial solution prefices $\mathcal{D}_{prompt} = \{(q_i, p_i)\}_{i\in[N]}$.
- **A completer** π_c that is a policy used for generating the full solution s given a question q and a partial solution prefix p. The generated full solution s contains p as the prefix and contain necessary rationales that lead to the final answer. The completer is not necessary an LLM policy and can also be a perfect oracle policy, or an LLM-based search policy.
- **A labeled dataset** constructed by using the completer π_c to find solution for the prompt dataset $\mathcal{D}_{prompt}$, i.e. $\mathcal{D}_{label} = \{(q_i, p_i, s_i, y_i)|s_i \sim \pi_c(\cdot|q_i, p_i), y_i = \text{Correct}(q_i, s_i)\}$ where s_i is a full solution generated by the completer π_c and y_i denotes the correctness label. The labels thus depend on the strength of the completer π_c.
- **A loss function** $\mathcal{L}$ for PRM training. By default, we use the logistic loss

$$\mathcal{L}(r_{process}, \mathcal{D}_{label}) = \mathbb{E}_{q_i,p_i,s_i,y_i\sim\mathcal{D}_{label}}[-y_i \log r_{process}(q_i, p_i) \tag{16}$$

$$- (1 - y_i) \log(1 - r_{process}(q_i, p_i))] \tag{17}$$

We first list some possible options for the completer,

- **A perfect oracle completer** π_{orcale}**.** We assume the existence of a perfect oracle completer π_{oracle} that is perfectly rational in the sense that, given a question q and a partial solution prefix p, π_{orcale} would make optimal reasoning towards the correct answer while also not driving any conclusions conflicting to the partial solution prefix when completing the rest reasoning trajectory. Therefore, if the solution prefix p contains any erroneous reasoning steps that conflict with the correct answer, π_{oracle} can not identify the correct answer and thus the label y for this pair (q, p) would be zero.
- **A base LLM completer** π_{base}**.** π_{base} is essentially an LLM that samples the rest of the solution given a question q and a partial solution prefix p. We assume a temperature of 1.0 and do not consider any sampling techniques such as top-p and top-k sampling. Therefore the label y is 1 with the probability of π_{base} sampling a correct solution.
- **A LLM-based search completer** π_{search}**.** We specially consider the search approach for automatic process label generation in Wang et al. (2024b), which tries to find a correct solution out of the sampled M solutions from a base LLM π_{base}. Clearly the search completer π_{search} is an augmented policy of π_{base}. The probability of π_{search} generating a correct solution is higher than the probability of π_{base} generating a correct solution.

Definition 1 (Value of a completer) *Given a question q and a solution prefix p, the value of a completer π_c is defined as*

$$V_{\pi_c}(q, p) = \mathbb{E}_{s\sim\pi(\cdot|q,p)}[Correct(q, s)] \tag{18}$$

Lemma 3 (PRM is learning a value function) *An optimal PRM that achieves the lowest PRM training loss in Eq. 16 is equivalent to the value function of the completer.*

Proof 3 *The optimal PRM $r^*_{process}(q_i, p_i)$ of PRM training loss in Eq. 16 is,*

$$\begin{aligned} r^*_{process}(q_i, p_i) &= Pr[y_i = 1 | s_i \sim \pi_c(q_i, p_i), y_i = Correct(q_i, s_i)] \\ &= Pr[Correct(q_i, s_i) | s_i \sim \pi_c(q_i, p_i)] \\ &= E_{s_i \sim \pi_c(\cdot|q_i, p_i)}[Correct(q_i, s_i)] \\ &= V_{\pi_c}(q_i, p_i) \end{aligned}$$

which is the same as the value of the completer π_c in Eq. 18.

Therefore PRM is actually learning the value of the completer π. In that sense, RL training that combines PR-Delta as dense rewards and the success reward is actually training the LLM to find correct solutions under the guidance of a stronger policy, i.e. the completer used to label the training data of the PRM.

Corollary 1 (PR-Delta guides LLM reasoning with a stronger policy) *Combining Lemma. 1 and Lemma. 3, PR-Delta actually guides the LLM to learn reasoning skills under the guidance of the stronger completer policy π_c, as shown in the policy gradient,*

$$\nabla_\theta J_r(\pi_\theta) = \mathbb{E}_{q \sim \mathcal{D}, s \sim \pi_\theta(\cdot|q)}[\sum_{k=1}^{K} \nabla_\theta \log \pi_\theta(s^{(k)}|q, p^{(k-1)}) \cdot (\alpha \cdot V_{\pi_c}(q, p^{(k)}) \cdot \mathbb{I}[k < K] \tag{19}$$

$$+ Correct(q, s) - \beta \cdot \sum_{k'=k}^{K} \log \frac{\pi_\theta(s^{(k')}|q, p^{(k'-1)})}{\pi_{ref}(s^{(k')}|q, p^{(k'-1)})})] \tag{20}$$

F Additional Case Studies of PRM

We evaluate the score of a open-sourced PRM 'ScalableMath/llemma-7b-prm-prm800k-level-1to3-hf'[3] on the following solution:

Question

What is the range of all the roots of $|x^2 - 6| = x$?

Solution

Step 1: Establish the equation by squaring both sides. We get $(x^2 - 6)^2 = x$.

PRM reward: 0.4874

Step 2: Expand the squared term. We get $(x^2 - 6)^2 = x^4 - 12x^2 + 36$.

PRM reward: 0.1667

Step 3: Set the expanded equation equal to x. We get $x^4 - 12x^2 + 36 = x$.

PRM reward: 0.4116

Step 4: Move all terms to one side to set the equation to zero. We get $x^4 - 12x^2 - x + 36 = 0$.

PRM reward: 0.7827

Step 5: Factor the polynomial. We can factor it as $(x^2 - 6)(x^2 + 1) = x$.

PRM reward: 0.0558

Step 6: Solve each factor for x. For $x^2 - 6 = x$:

$$x^2 - x - 6 = 0$$

$$(x - 3)(x + 2) = 0$$

[3] https://huggingface.co/ScalableMath/llemma-7b-prm-prm800k-level-1to3-hf

So, the solutions are $x = 3$ and $x = -2$. For $x^2 + 1 = x$:

$$x^2 - x + 1 = 0$$

This quadratic equation has no real roots because its discriminant is negative:

$$\Delta = (-1)^2 - 4 \cdot 1 \cdot 1 = 1 - 4 = -3$$

PRM reward: 0.4244

Step 7: Determine the range of the roots. The roots we found are $x = 3$ and $x = -2$.

PRM reward: 0.6234

Step 8: Identify the minimum and maximum values of the roots. The minimum value is -2 and the maximum value is 3. Therefore, the range of all the roots of $|x^2 - 6| = x$ is from -2 to 3. The answer is $\boxed{[-2, 3]}$.

PRM reward: 0.4244