

Good Learners Think Their Thinking: Generative PRM Makes Large Reasoning Model More Efficient Math Learner

Tao He^{1*} Rongchuan Mu^{1*} Lizi Liao² Yixin Cao³ Ming Liu^{1†} Bing Qin¹

¹ Harbin Institute of Technology

² Singapore Management University

³ Fudan University

Corresponding to: the@ir.hit.edu.cn

Abstract

Large reasoning models (LRMs) have recently shown promise in solving complex math problems when optimized with Reinforcement Learning (RL). But conventional approaches rely on outcome-only rewards that provide sparse feedback, resulting in inefficient optimization process. In this work, we investigate the function of process reward models (PRMs) to accelerate the RL training for LRMs. We propose a novel intrinsic signal-driven generative process evaluation mechanism operating at the thought level to address major bottlenecks in RL-based training. Specifically, instead of requiring PRMs to know how to solve problems, our method uses intrinsic signals in solutions to judge stepwise correctness and aggregate contiguous correct/incorrect steps into coherent ‘thought’ units. This structured, thought-level rewards enable more reliable credit assignment by reducing ambiguity in step segmentation and alleviating reward hacking. We further introduce a capability-adaptive reward mechanism that dynamically balances exploration and exploitation based on the LRM’s current proficiency, guiding learning without stifling creative trial-and-error. These innovations are integrated into a new off-policy RL algorithm, **TP-GRPO**, which extends grouped proximal optimization with process-based rewards and improves training efficiency. Experiments on 1.5B and 7B parameter LRMs demonstrate that our method achieves higher problem-solving accuracy with significantly fewer training samples than outcome-only reward baselines. The results validate that well-structured process rewards can substantially accelerate LRM optimization in math reasoning tasks. Code is available at https://github.com/cs-holder/tp_grpo.

1 Introduction

The remarkable success of Reinforcement Learning with Verifiable Rewards (RLVR) methods [Guo et al., 2025, Liu et al., 2025a, Hu et al., 2025], exemplified by DeepSeek R1 [Guo et al., 2025] and Kimi k1.5 [Team et al., 2025b], demonstrates the effectiveness of Reinforcement Learning (RL) in enhancing the reasoning capabilities of Large Language Models (LLMs). At their core, these methods scale RL by employing efficient and model-free rule-based outcome reward functions [Shao et al., 2024, Yue et al., 2025, Xu et al., 2025]. It is well established that RL revolves around two fundamental challenges: exploration and exploitation [Berger-Tal et al., 2014, Wang et al., 2018, Fruit, 2019, Hao et al., 2025]. RLVR-style approaches exploration by leveraging efficient reward

* Equal contribution.

† Corresponding author.

functions to enable large-scale exploration, akin to how learners improve problem-solving skills through extensive practice without retrospect their thinking processes.

However, such methods focus solely on the correctness of final answers and neglect to exploit the informative value of intermediate reasoning trajectories [Chen et al., 2025a, Zou et al., 2025, Chen et al., 2024a, Ye et al., 2025]. In contrast, skilled learners reflect on their redundant thinking after exercise, engaging in selective learning to maximize the learning efficiency. This strategy inspires the use of process rewards to guide the learning of Language Reasoning Models (LRMs). Motivated by this analogy, we want to explore a central question: **can well-designed process rewards enhance the learning efficiency of LRMs?** We hope to motivate a broader discussion on the significance of process rewards for LRM training.

Prior to DeepSeek R1 [Guo et al., 2025], learning discriminative process reward models (PRMs) had already received considerable attention. OpenAI released a public dataset for process evaluation [Lightman et al., 2023]; Wang et al. [2023b, 2024c] proposed a Monte Carlo Tree Search-based method to automatically annotate process rewards; and Qwen’s technical report explored the development of Process Reward Models [Zhang et al., 2025] and offered a benchmark for measuring the ability to assess the quality of the reasoning trace [Zheng et al., 2024b]. Nevertheless, DeepSeek R1 revealed the limitations of discriminative PRMs, including difficulty in step segmentation, limited generalization, vulnerability to reward hacking, and training inefficiencies [Guo et al., 2025]. Therefore, some recent efforts have shifted toward Generative PRMs (GenPRMs) [Khalifa et al., 2025, Feng et al., 2025, She et al., 2025], which leverage powerful LRMs to evaluate reasoning processes with detailed thinking [Liu et al., 2025b, Chen et al., 2025b].

In light of existing critiques of PRMs [Guo et al., 2025, Song et al., 2025, Zheng et al., 2024a], we argue that a robust GenPRM-based process evaluation mechanism should satisfy three essential criteria: **1) Appropriate unit segmentation.** Excessively fine-grained evaluation increases the burden of process evaluation, thus amplifying the risk of flawed evaluation. And excessive coarseness in segmentation can also result in correct and incorrect steps being grouped together. **2) Evaluation capability decoupled from reasoning proficiency.** Current GenPRM approaches often assess step correctness via thinking mode [Feng et al., 2025], which implicitly assumes the PRM can solve the original problem. This strong prerequisite elevates the reasoning ability demands on the GenPRM, limiting its reliability when facing difficult questions. **3) Balanced exploration and exploitation.** Overemphasizing process rewards may impair the LRM’s exploratory capabilities, like trial-and-error reasoning, reflection, and self-correction. Therefore, it is crucial to mitigate the negative impact of process rewards on exploration while leveraging their benefits for guided learning.

Grounded in these criteria, we design a new GenPRM-based process evaluation mechanism, incorporating three principal innovations compared to existing methods: **1)** We propose a thought-level evaluation strategy that aggregates contiguous correct or incorrect reasoning steps into coherent evaluation units, termed thoughts. This design anchors reward assignment in the correctness rather than the semantic granularity of step segmentation, thereby reducing sensitivity to inconsistent step definitions. **2)** We introduce an intrinsic signal-driven evaluation paradigm that significantly lowers the reliance on the reasoning capabilities of the GenPRM. Unlike prior GenPRMs that require strong multi-step reasoning abilities, our method leverages intrinsic signals within reasoning trajectories for process evaluation, relying primarily on comprehension of the reasoning process, thus relaxing the capability requirements on the reward model. **3)** We develop a capability-adaptive reward computation mechanism that dynamically calibrates reward signals based on the current reasoning proficiency of the LRM. This mechanism facilitates a better balance between exploration and exploitation during RL training, while maintaining focus on critical segments relevant to the final answer. Crucially, it enables guided learning with greater robustness against reward hacking. Built upon this enhanced process evaluation framework, we modify the GRPO [Shao et al., 2024] and propose our TP-GRPO (Thought-level Process-reward-based GRPO) algorithm. Considering the inefficiencies of GenPRM, we further design an efficient off-policy training pipeline that mitigates GPU idling caused by the generative process evaluation.

We validate our approach on the DeepSeek-R1-Distill-Qwen family of models [Guo et al., 2025]. Our 1.5B model, trained on only 700 problems, achieves +4.32% improvements in Pass@1 on the AIME 2024 benchmark [Maxwell-Jia, 2024]; with 1800 training problems, the improvements increase to +5.98%. Remarkably, the 7B model, trained on just 1070 problems, also achieves +6.67% gain in Pass@1, substantially outperforming baselines trained with outcome-only rewards. These results

strongly support our central hypothesis: that the judicious introduction of process-level rewards can significantly accelerate learning in math reasoning tasks.

Our main contributions are summarized as follows:

- We propose a new GenPRM paradigm that advances existing approaches along three dimensions: evaluation granularity, dependency on reasoning capabilities, and reward formulation. Our design addresses key limitations such as ambiguous process segmentation, over-reliance on reasoning strength, and reward hacking.
- We integrate the proposed process reward mechanism with GRPO and introduce a novel RL algorithm, TP-GRPO. To address the efficiency bottlenecks introduced by GenPRM, we further develop a scalable off-policy training pipeline.
- Experiments on DeepSeek-R1-Distill 1.5B and 7B models demonstrate the effectiveness and efficiency of our approach. The performance gains empirically support our core hypothesis: integrating well-designed process rewards significantly enhances learning efficiency over outcome-only supervision GRPO.

2 Preliminary

2.1 RL Foundation: GRPO

Our method builds on GRPO, which estimates advantage values via Monte Carlo (MC) sampling, thereby avoiding eliminating the need to learn a value function. Specifically, given a query q , GRPO samples G solutions $o_{i=1}^G$ from the old policy $\pi_{\theta_{old}}$ and optimizes the current policy π_{θ} using:

$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E}[q \sim P(Q), \{o_i\}_{i=1}^G \sim \pi_{\theta_{old}}(Q|q)]$$

$$\frac{1}{\sum_{i=1}^G |o_i|} \sum_{i=1}^G \sum_{t=1}^{|o_i|} \left\{ \min \left[\rho_{i,t} \hat{A}_{i,t}, \text{clip}(\rho_{i,t}, 1 - \epsilon_{\text{low}}, 1 + \epsilon_{\text{high}}) \hat{A}_{i,t} \right] - \beta D_{\text{KL}}[\pi_{\theta} \parallel \pi_{\text{ref}}] \right\}, \quad (1)$$

where $\rho_{i,t} = \frac{\pi_{\theta}(o_{i,t}|q, o_{i,<t})}{\pi_{\theta_{old}}(o_{i,t}|q, o_{i,<t})}$ refers to the policy ratio, and $\hat{A}_{i,t}$ denotes the estimated advantage for the t -th token of solution o_i . Here, we adopt two effective improvements from DAPO over GRPO, including the use of token-level loss instead of solution-level loss, as well as the application of distinct clipping ranges, ϵ_{low} and ϵ_{high} . Depending on the availability of process-level rewards, GRPO supports two variants of advantage estimation:

- **With Outcome-only Rewards.** Motivated by the dominant performance of DeepSeek R1, most RLVR research works rely solely on rule-based outcome rewards. For each sampled solution o_i , a scalar outcome reward r_i^o is assigned. Rewards in a group are normalized as:

$$\hat{r}_i^o = \frac{r_i^o - \text{mean}(\{r_j^o\}_{j=1}^G)}{\text{std}(\{r_j^o\}_{j=1}^G)} \quad (2)$$

In this case, the advantage of each token is set uniformly as $A_{i,t} = \hat{r}_i^o$.

- **With Process Rewards.** When intermediate step-level rewards are accessible, the computation of the advantage differs accordingly. Let $R = \{r_1^{\text{index}(1)}, \dots, r_1^{\text{index}(K_1)}, \dots, r_G^{\text{index}(1)}, \dots, r_G^{\text{index}(K_G)}\}$ denote the full set of rewards across all G solutions, where $\text{index}(k)$ denotes the position of the last token in the k -th step. GRPO first normalizes all step rewards:

$$\hat{r}_i^{\text{index}(j)} = \frac{r_i^{\text{index}(j)} - \text{mean}(R)}{\text{std}(R)} \quad (3)$$

and then computes the advantage of token t in o_i as the cumulative sum of normalized rewards from all steps ending at or after the t -th token:

$$A_{i,t} = \sum_{\text{index}(j) \geq t} \hat{r}_i^{\text{index}(j)} \quad (4)$$

Previous attempts to utilize process rewards often relied on learning a specialized process reward model, which suffers from challenges like reduced accuracy and vulnerability to reward hacking. As a result, interest in process rewards declined after the emergence of strong outcome-based baselines. In contrast, our work avoids explicitly training the process reward model. Instead, we leverage the outstanding reasoning capability of LRM to evaluate intermediate steps.

2.2 Problem Formulation

Given a math problem x , the LRM π_θ generates a Long CoT o , which typically consists of two parts: *think* and *answer* Guo et al. [2025]. The *think* part records the full reasoning trace, often including multiple thoughts after a series of trial and error and reflection. The *answer* part is usually a concise summary based on the preceding *think*, **containing only the main reasoning thought utilized in the think that leads to the final answer**. Assuming consistency between *answer* and *think*, we argue that improving reasoning quality primarily hinges on refining the *think*. Therefore, our process evaluation focuses on the *think* part instead of the *answer* one. For convenience, we will abuse the term *solution* to represent *think* in the following sections unless otherwise specified.

3 Our Method

In this section, we propose a new RL algorithm driven by process rewards. We first introduce a novel evaluating mechanism for intermediate reasoning steps in math problem solving §(3.1). Based on this evaluation mechanism, we further develop an RL algorithm that effectively leverages these process rewards §(3.2). Finally, we present an efficient off-policy training framework §(3.3). An overview of the entire method is illustrated in Figure 1.

3.1 Intrinsic Signal-driven Thought-level Generative Process Evaluation

Before performing process evaluation, we decompose each solution into a sequence of steps with suitable granularity. While we prompt the LRM to perform such decompositions, alternative approaches are equally applicable, as our ultimate goal is to perform evaluation at the thought level.

Then, we propose a systematic three-stage framework for evaluating reasoning processes. The 1st stage introduces a tailored prompting pipeline to assess the correctness of each reasoning step, with reduced reliance on the evaluator’s reasoning capabilities. The 2nd stage aggregates step-level evaluations into thought-level assessments, effectively mitigating challenges arising from ambiguous step segmentation and noisy judgments. Finally, the 3rd stage incorporates a capacity-adaptive reward mechanism that assigns fine-grained rewards based on the correctness of each reasoning thought. An overview of the evaluation pipeline is provided in Figure 1.

3.1.1 Stage 1: Intrinsic Signal-driven Step-level Process Evaluation

Similar to existing generative PRMs, our approach starts with step-level evaluation. Prior work has explored training specialized generative PRMs or prompting LRM to directly assess the correctness of each reasoning step. However, these methods implicitly assume that the LRM-based PRM must possess the reasoning capability to solve the problem—a requirement that significantly increases the difficulty of process evaluation. We introduce a new evaluation paradigm based on intrinsic signals in Long CoT. The PRM assesses reasoning steps mainly through comprehension rather than reasoning, drastically lowering the capability requirements for PRM. Based on the correctness of solutions, we design two distinct evaluation procedures to reflect different expectations for process rewards:

Case 1: Correct solutions. We first aim to distinguish which steps within a correct solution are valid and which correspond to erroneous trials. To address this, we decompose the evaluation into three simpler subtasks, proposing a 3-step evaluation pipeline.

- 1) Prompt the PRM to identify the specific step where an error is recognized through reflection. This is relatively easy as the solution often includes intrinsic reflective triggers such as “wait” or “I made a mistake”;
- 2) Prompt the PRM to identify the earliest step responsible for this error—referred to as the error cause. Since the reflection thought in 1) often includes an analysis of the error cause, pinpointing this step is generally not difficult;

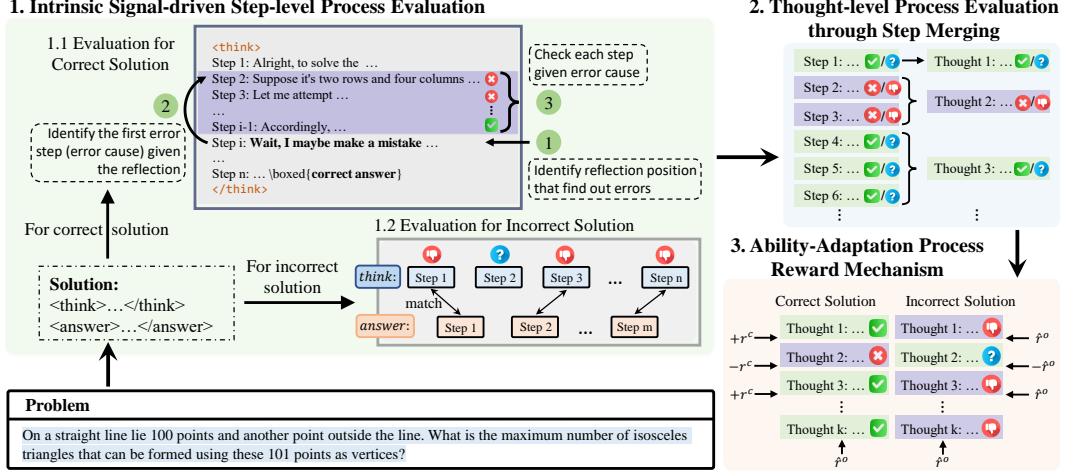


Figure 1: The pipeline of intrinsic signal-driven thought-level generative process evaluation. The whole process consists of three steps: (1) Intrinsic Signal-driven Step-level Process Evaluation, which evaluates each reasoning step using intrinsic signals from solutions; (2) Thought-level Process Evaluation through Step Merging, which merges continuous steps with the same correctness to enable thought-level evaluation; and (3) Ability-Adaptation Process Reward Mechanism, which calculates rewards based on model capability, balancing exploration and exploitation.

- 3) Prompt the PRM to evaluate the correctness of each intermediate step between the error cause and the reflection step. Given both the error span and its reason, evaluating the correctness of each step becomes more manageable.

This process naturally mirrors a learner’s review and reorganization of reasoning solutions after problem-solving. **Besides, to avoid suppressing cognitive behaviors such as reflection and self-correction during RL training, we treat the reflective steps as correct in Step 3).**

Case 2: Incorrect solutions. For incorrect solutions, the evaluation strategy for correct solutions is no longer applicable, as the reflective steps in such cases may be flawed—otherwise, the solution would be corrected. Instead, we propose a matching-based evaluation approach. Specifically, we compare each step in the *think* with the *answer* and identify matched *think* steps. Our core assumption is that while the *think* may contain various exploratory reasoning paths, the one ultimately applied in the *answer* must be incorrect, as the answer is incorrect.

By performing fine-grained, step-level matching between *think* and *answer*, we can label the matched steps as flawed, while unmatched steps are marked as uncertain. This matching-based evaluation mechanism significantly reduces the reliance on the evaluator’s reasoning capabilities—requiring only the ability to understand a given reasoning trace rather than solving the math problem directly. In fact, we do not attempt to assess the precise correctness of every individual step, as such fine-grained evaluation imposes high reasoning demands on PRMs. Conversely, treating all steps in an incorrect solution as invalid—typical in outcome-only reward approaches—risks excessive penalization. Therefore, we pursue a principled trade-off: enabling selective penalization (compared to outcome-only methods) while significantly reducing the reasoning capability required (compared to accurate evaluation for each step). As a result, our method remains effective and generalizable even when high-capacity reasoning models are not available. All detailed prompts used are provided in Appendix D.

3.1.2 Step 2: Thought-level Process Evaluation through Step Merging

By merging the step-level evaluation results, we further obtain thought-level process evaluation. The implementation is conceptually simple: for correct solutions, consecutive correct or incorrect reasoning steps are merged into correct or incorrect units, named as ‘**thoughts**’, respectively; for incorrect solutions, consecutive steps that are matched or unmatched with the *answer* are grouped into matched or unmatched thoughts. Instead of assigning per-step rewards, we assign a single reward

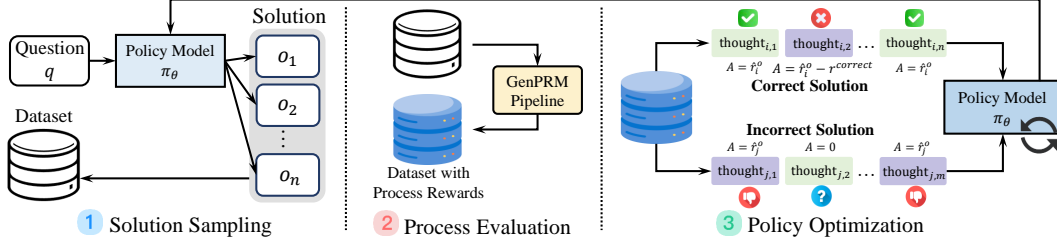


Figure 2: The overview of our proposed efficient off-policy training pipeline. This design ensures full utilization of GPU resources at each stage—all GPUs are either concurrently used for sampling, process evaluation, or parameter updates, avoiding any idle computation.

at the end of each thought segment, as detailed in Stage 3. Our thought-level evaluation offers the following advantages:

- **Clearer process segmentation:** Thoughts are split based on correctness, yielding more coherent and interpretable segments compared to ambiguous step-level divisions.
- **Mitigating noisy reward signals.** Given the limited accuracy of PRMs and the typically large number of reasoning steps in Long CoT, evaluating at the thought level significantly reduces the impact of noisy evaluations.
- **Resistance to reward hacking:** A common reward hacking issue arises when the LRM obtains more reward by repeatedly generating positive patterns. Merging homogeneous steps prevents such behavior by withholding additional reward for repeated positive evaluations.
- **Structured segmentation for RL.** Through step merging, solutions are segmented into alternating correct and incorrect thoughts or alternating matched and unmatched thoughts—forming a structured basis for developing RL algorithms that better exploit process-level feedback, as further detailed in section 3.2.

3.1.3 Step 3: Ability-Adaptation Process Reward Mechanism

After performing process evaluation, we further design two distinct process reward mechanisms based on the solution correctness.

For Correct solutions. For a correct solution o_i , we assign the same positive reward $+r^{\text{correct}}$ to all correct thoughts and the same negative reward $-r^{\text{correct}}$ to all incorrect ones.

The design of r^{correct} needs to strike a balance between exploration and exploitation. When a problem is hard for the current LRM, we make rewards primarily stem from the outcome by keeping r^{correct} small, thereby prioritizing encouraging exploration, including more trial and error. In contrast, once the LRM can reliably solve the problem, the focus shifts toward refining and consolidating the reasoning process. And we assign larger process rewards to penalize erroneous reasoning, thereby enhancing learning efficiency. To achieve the above targets, we define r_{correct} as:

$$r^{\text{correct}} = \alpha \cdot \text{acc_mean} \quad (5)$$

where $\alpha > 0$ is hyper-parameter to control the strength of process rewards, and acc_mean is the average accuracy of G solutions within the same group sampled in GRPO Shao et al. [2024]. acc_mean reflects the LRM’s capability on the current problem. When all solutions are incorrect, $r^{\text{correct}} = 0$ and the setup reduces to the Outcome-only GRPO; when all solutions are correct, $r^{\text{correct}} = \alpha$ and the strongest process reward is applied to reinforce or suppress specific thoughts. We discuss the optimization implications of this design in Section 3.2.

For Incorrect solutions. For an incorrect solution o_i , we penalize only the reasoning thoughts in *think* directly responsible for the error—i.e. thoughts matched with the *answer*. Matched thoughts are penalized with $+r^{\text{incorrect}}$ at their final token, while unmatched thoughts receive $-r^{\text{incorrect}}$. We define incorrect reward $r_i^{\text{incorrect}}$ as:

$$r_i^{\text{incorrect}} = \hat{r}_i^o \quad (6)$$

where \hat{r}_i^o is the normalized outcome reward of solution o_i defined in Eq. 2. This ensures that thoughts unmatched to *answer* are not penalized, thereby avoiding the suppression of potentially useful but not fully explored reasoning thoughts Wang et al. [2025]. We further explain this reward design in Section 3.2.

3.1.4 Limitations

The preceding section outlines our proposed process evaluation framework. While it introduces several novel mechanisms, certain limitations remain:

- The primary limitation lies in the 1st-stage step-level evaluation. Due to its inherently multi-step procedure, this stage exhibits considerable instability—an issue commonly observed in GenPRM approaches that lack dedicated task-specific training. To address this, future work will explore the design of more robust and instruction-sensitive prompts, as well as the fine-tuning of GenPRMs to enhance evaluation reliability and consistency.
- The proposed process evaluation method requires multiple calling for the GenPRM, resulting in low evaluation efficiency. This is also a common limitation for previous GenPRMs. To mitigate this issue, we introduce an off-policy training pipeline introduced in the following that minimizes GPU idle by decoupling sampling and optimization. Additionally, we employ heuristic rules to filter solutions committed to process evaluation—such as those with excessively short *think* or *answer* content—thereby improving overall efficiency.
- Moreover, for incorrect solutions, our approach does not exactly verify the correctness of each intermediate step. Instead, it penalizes all tokens within the primary thought deemed responsible for the final error. It inevitably results in misattribution of penalties to partially correct reasoning steps in the primary thought, as not all steps within the primary thought are erroneous. We regard this as a pragmatic compromise between two extremes: coarse-grained outcome-level rewards, which indiscriminately penalize all tokens, and fine-grained step-level rewards, which require the high capacity of precise error localization. This trade-off balances the evaluation accuracy with the practical constraints on the PRM’s capacity.

3.2 TP-GRPO: GRPO based on Thought-level Process Reward

Finally, we integrate the proposed process evaluation framework with GRPO [Guo et al., 2025]—the most widely adopted RL algorithm in current LLM training—and introduce a new algorithm, TP-GRPO. TP-GRPO jointly optimizes the LLM under the guidance of both process and outcome rewards. For outcome rewards, we retain the standard group-level normalization as in Eq. 2. For correct solutions o_i , we assign $+r^{\text{correct}}$ to the final token for each correct thought and $-r^{\text{correct}}$ to that of each incorrect thought. For incorrect solutions, we assign $+r_i^{\text{incorrect}}$ to thoughts matched with the answer, and $-r_i^{\text{incorrect}}$ to unmatched thought segments. The final thought segment receives no process reward, as its last token already receives the outcome reward. In summary, we formalize the reward assignment as follows:

$$r_{i,t} = \begin{cases} \hat{r}_i^o, & o_{i,t} \text{ is the final token of solution } o_i \\ +r^{\text{correct}}, & o_i \text{ is correct } \wedge o_{i,t} \text{ is the end token of a correct thought} \\ -r^{\text{correct}}, & o_i \text{ is correct } \wedge o_{i,t} \text{ is the end token of an incorrect thought} \\ +r_i^{\text{incorrect}}, & o_i \text{ is incorrect } \wedge o_{i,t} \text{ is the end token of a matched thought} \\ -r_i^{\text{incorrect}}, & o_i \text{ is incorrect } \wedge o_{i,t} \text{ is the end token of an unmatched thought} \\ 0, & \text{else} \end{cases} \quad (7)$$

Then, the advantage of each token $o_{i,t}$ is calculated as $A_{i,t} = \sum_{t'=t}^{|o_i|} r_{i,t'}$ following the process supervised GRPO [Shao et al., 2024], where $|o_i|$ is the number of tokens in o_i . Finally, the optimization objective is defined in Eq. 1:

To better understand the design of these two types of process rewards, we establish the following theoretical properties from the perspective of advantage estimation in GRPO:

Proposition 1. *For a correct solution o_i , token $o_{i,t}$ in correct thoughts have advantage $A_{i,t} = \hat{r}_i^o$; while token $o_{i,t}$ in incorrect thoughts have advantage $A_{i,t} = \hat{r}_i^o - r^{\text{correct}}$.*

Implication: This proposition shows that tokens within correct thoughts retain the same advantage as using outcome-only rewards, preserving their original optimization objective during process-reward-augmented RL training. In contrast, tokens within incorrect thoughts receive reduced advantage values compared with using outcome-only rewards, effectively attenuating the optimization of flawed reasoning.

Proposition 2. *For an incorrect solution o_i , token c_t in matched thoughts have advantage $A_{i,t} = \hat{r}_i^o$, while token c_t in unmatched thoughts have advantage $A_{i,t} = 0$.*

Implication: This proposition demonstrates that tokens within matched thoughts retain the same advantage values as under outcome-only rewards, thereby achieving the objective also mentioned above: introducing process rewards without altering the original training objective for these incorrect tokens during RL training. In contrast, unmatched thoughts receive 0 advantage, avoiding unintended penalization since their correctness remains uncertain.

The proofs of these two propositions are presented in Appendix A.

The importance of preserving the original optimization objective. A widely discussed challenge in integrating process rewards is reward hacking, which we attribute to distortions in the advantage estimation induced by process rewards. While the ultimate goal in math reasoning is to arrive at the correct final answer, process rewards may inadvertently distort the LRM’s optimization objective. Specifically, LRM may exploit the process reward to inflate the overall advantage estimate, leading the optimization objective away from deriving the correct answer—manifesting, for example, in the repeated generation of steps that yield high process rewards without making meaningful progress toward the solution.

Our reward design explicitly avoids this pitfall. Taking a correct solution as an example, TP-GRPO suppresses error thoughts without altering the advantage of tokens within the correct thoughts. As a result, the optimization objective for these tokens in key thoughts (correct or matched) remains aligned with that under the outcome reward alone—namely, improving reasoning accuracy—thus preventing any distortion from the ultimate learning objective.

3.3 Efficient Off-Policy Training Pipeline

Prompting LLMs for process evaluation introduces significant efficiency challenges, as it requires multiple inference calls to the GenPRM. This inevitably leads to substantial GPU idle time under an on-policy training paradigm. To address this issue, we design an off-policy training pipeline that improves GPU utilization and accelerates training. This pipeline consists of three stages within each iteration:

1. **Stage 1: Solution Sampling.** We deploy the latest updated actor LRM using all GPUs to sample candidate solutions for each problem. Suppose Stage 3 performs S training steps, each with M problems, and each problem generates G candidate solutions. Then, Stage 1 produces a total of $S \times M \times G$ solutions for $S \times M$ problems.
2. **Stage 2: Process Evaluation.** We deploy multiple LLMs using all GPUs as the evaluator and evaluate the solutions using a multi-process, multi-threaded setup. This stage computes process rewards for $S \times M \times G$ sampled solutions.
3. **Stage 3: Policy Optimization.** We train the actor LRM via the RL algorithm using all GPUs, based on the solutions and their corresponding process and outcome rewards from Stage 1 and 2. We filter out solutions without annotated process rewards after Stage 2 to focus on process rewards.

4 Experiments

In this section, we evaluate the effectiveness of TP-GRRO in improving LRMs’ math reasoning. Beyond accuracy assessment, we further examine training efficiency to validate that our process reward formulation improves the exploitation of reasoning trajectories. Given computational limitations, we focus our experiments on smaller LRMs. To meet the format requirement that explicitly includes both *think* and *answer* components, we select the DeepSeek-R1-Distill-Qwen 1.5B and 7B models for experiments. We leave broader studies across more scales and variants to future work.

Model	AIME 24 Avg@16	AIME 25 Avg@16	AMC 23 Avg@16	MATH-500 Avg@16	Olympiad Avg@16	Avg.
Base Model	28.80	22.50	62.90	82.80	43.30	48.06
Outcome Reward-based RLVR methods						
AutoThink (Stage 1)	28.13	19.58	60.39	79.74	40.44	45.66
Open-RS1	30.42	20.83	63.78	83.01	43.91	48.39
Still-3-1.5B-Previous	32.50	23.33	66.94	84.19	45.96	50.58
AdaptThink	33.75	24.38	69.35	84.18	45.04	51.34
AReal-1.5B-Preview(Stage 1)	30.83	23.75	66.64	84.33	45.51	50.21
DeepScaler-1.5B-Preview	43.10	29.38	73.60	87.80	50.00	56.78
GRPO Replication (850 steps)	32.71	24.58	64.53	82.70	43.67	49.64
Process Reward-based methods						
TP-GRPO (140 steps)	33.12	25.63	64.01	83.81	43.91	50.10

Table 1: Overall performance based on DeepSeek-R1-Distill-Qwen-1.5B.

4.1 Experimental Settings

Baselines We primarily compare TP-GRPO with the reproduction of GRPO using only outcome rewards, aiming to demonstrate the advantage of our proposed process rewards in terms of both performance and efficiency. Specifically, we evaluate whether TP-GRRO can exceed the performance of outcome-only RL methods while requiring fewer training prompts, thereby validating its efficiency. During TP-GRRO training, to minimize the influence of outcome rewards, we exclude solutions that do not obtain any intermediate rewards after process evaluation. Besides, we also include results from several open-source baselines trained on DeepSeek-R1-Distill-Qwen models as reference and also compare efficiency with them.

Training Data and Evaluation Benchmark We conduct all experiments on DeepScaler-40K Luo et al. [2025]. Due to limitations in computational resources, we do not train with full dataset. We follow prior work and evaluate on 5 widely-used math benchmarks: AIME 24 [Maxwell-Jia, 2024], AIME 25 [math ai, 2025], AMC 23 [Li et al., 2024a], MATH-500 [Hendrycks et al., 2021], and Olympiad [He et al., 2024]. For experiments with DeepSeek-R1-Distill-1.5B, we perform 16 independent Pass@1 evaluations and report the average (Avg@16) as final results. Due to the higher inference cost of DeepSeek-R1-Distill-7B and limited sizes of AIME 24, AIME 25, and AMC 23, we report Avg@5 on these subsets and Pass@1 on MATH-500 and Olympiad. Besides, we aim to further investigate improvements in training efficiency. Due to variations in RL frameworks (e.g., TRL [von Werra et al., 2020] and VeRL [Sheng et al., 2025]) and training settings, different methods are not directly comparable in terms of step number. To ensure a fairer comparison, we design a new metrics *Effic.* to measure the training efficiency: $Effic. = \frac{\text{Improvement}}{\# \text{training solutions}} * 10^5$.

Hyperparameters All training experiments are conducted using the TRL framework and vLLM backend. We use a batch size of 5, a fixed learning rate of $1e-6$, and sample 8 rollouts per prompt. For TP-GRPO, we set the KL loss coefficient $\beta = 0$ following previous practice. The reproduced GRPO also uses the token-level loss and $\epsilon_{\text{high}} = 0.28$ setting introduced in DAPO [Yu et al., 2025]. For on-policy experiments, we save checkpoints every 50 training steps and perform evaluations accordingly. For off-policy experiments, we first collect solutions for 50 training steps at each iteration, then perform unified solution sampling and process evaluation. More training and evaluation settings can be found in the Appendix B.

4.2 Main Results

Tables 1 and 2 present the experimental results using DeepSeek-R1-Distill-Qwen-1.5B and DeepSeek-R1-Distill-Qwen-7B as base models, respectively. First, TP-GRPO achieves consistent improvements over the base models: on the 1.5B model, it attains an average gain of +2.04 after training on only 700 prompts (140 steps), with particularly strong improvements of +4.32 and +3.13 on AIME 2024 and AIME 2025, respectively. On the 7B model, TP-GRPO achieves an average improvement of +3.43 using just 1070 prompts (214 steps), including gains of +6.67 and +6.66 on AIME 2024 and

Model	AIME 24 Avg@5	AIME 25 Avg@5	AMC 23 Avg@5	MATH-500 Pass@1	Olympiad Pass@1	Avg.
Base Model	54.00	36.67	81.20	92.00	55.11	63.80
Outcome Reward-based RLVR methods						
AutoThink (Stage 1)	52.67	32.67	77.59	90.00	54.07	61.40
ReasonFlux-F1	54.67	34.00	80.00	91.60	55.26	63.11
AdaptThink	52.67	38.00	82.17	90.80	56.89	64.11
AReal-boba-RL-7B	60.00	44.67	86.02	93.60	60.89	69.04
Skywork-OR1-7B	69.33	51.33	87.71	93.40	61.33	72.62
GRPO Replication (400 steps)	58.67	38.00	81.20	92.40	56.44	65.34
Process Reward-based methods						
TP-GRPO (214 steps)	60.67	43.33	83.13	92.00	57.04	67.23

Table 2: Overall performance based on DeepSeek-R1-Distill-Qwen-7B.

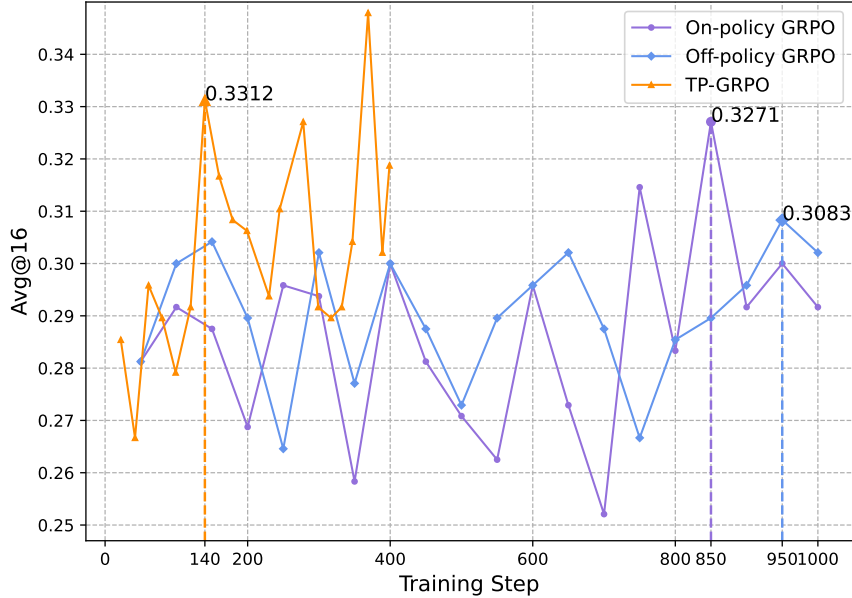


Figure 3: The trend of accuracy (Avg@16) on AIME 2024 over training steps for DeepSeek-R1-Distill-Qwen-1.5B base model.

AIME 2025. These results validate the efficacy of our thought-level process evaluation mechanism in guiding effective policy updates with limited data. Moreover, TP-GRPO outperforms our reproductive on-policy GRPO baseline, which relies solely on outcome rewards. On the 1.5B model, TP-GRPO trained on 700 prompts surpasses the performance of on-policy GRPO trained on 4250 prompts. Similarly, on the 7B model, TP-GRPO with 1,070 prompts significantly exceeds the performance of on-policy GRPO after its 400 training steps. This substantial gain in learning efficiency highlights the advantage of incorporating structured process feedback.

A more intuitive comparison is provided in Figure 3, which plots the trend of reasoning accuracy (Avg@16) on AIME 2024 over training steps. Here, off-policy GRPO denotes a GRPO variant using the off-policy training paradigm but without process rewards. All baseline methods are evaluated every 50 training steps. In contrast, TP-GRPO exhibits a denser checkpoint schedule. This arises because, although TP-GRPO samples enough solutions per iteration to support 50 update steps, only a subset receives non-zero process rewards after evaluation. To more precisely analyze the role of process rewards in training, we filter out those solutions lacking process rewards. As a result, the effective number of training samples available in the 3rd stage of each iteration is reduced, which in

Model	Traing Steps	Batch Size	Rollout Size	<i>Effic.</i>
DeepSeek-R1-Distill-Qwen-1.5B				
AutoThink (Stage 1)	220	128	16	-0.53
AdaptThink	300	128	16	0.53
Open-RS1	100	96	6	0.57
Still-3-1.5B-Previous	250	128	8	0.98
AReaL-1.5B-Preview(Stage 1)	200	1024	8	0.13
DS-1.5B-Preview	1750	128,128,128	8,16,16	<0.49
GRPO Replication	850	5	8	4.65
TP-GRPO	140	5	8	36.43
DeepSeek-R1-Distill-Qwen-7B				
AutoThink (Stage 1)	220	128	16	-0.53
ReasonFlux-F1	3750	16	1	-1.15
AdaptThink	150	128	16	0.10
AReaL-boba-RL-7B	250	512	16	0.37
Skywork-OR1	1320	256,160	16,32	<0.26
GRPO Replication	400	5	8	9.6
TP-GRPO	214	5	8	40.07

Table 3: Training hyperparameters for efficiency comparison.

turn limits the number of update steps that can be performed and leads to more frequent evaluation intervals. As shown in Figure 3, TP-GRPO exhibits a significantly steeper learning curve, indicating faster convergence and higher training efficiency. This provides strong evidence that TP-GRPO enhances the exploitation of reasoning trajectories through well-designed process rewards, enabling efficient learning to strengthen the LRM’s math reasoning capabilities. It should be noted that, for TP-GRPO, although a higher peak performance is observed after 140 steps, we find that the corresponding checkpoint shows limited improvement on AIME 2025. Considering overall training efficiency and cross-dataset generalization, we select the checkpoint saved at step 140 as the final TP-GRPO result.

4.2.1 More Efficiency Comparison

We further compare TP-GRPO with recent strong open-source RLVR methods (see Tables 1 and 2). Although our method lags behind the SOTA models in terms of absolute performance, this gap is primarily attributed to differences in the scale of training data used. RLVR approaches achieve RL scaling through extensive and efficient exploration. To enable a fairer comparison, Table 3 summarizes the training settings of all methods with respect to the efficiency metric (*Effic.*). Notably, TP-GRPO achieves the highest *Effic.*, outperforming all open-source baselines in terms of learning efficiency. This result further demonstrates the advantage of TP-GRPO in improving training efficiency and exploitation for reasoning trajectories.

Model	AIME 24	AIME 25	Training Steps
Base Model	28.80	22.50	-
TP-GRPO	33.12	25.63	140
- w/o CS Reward	30.00	25.00	58
- w/o IS Reward	32.01	22.29	90

Table 4: Ablation study on process rewards from correct vs. incorrect solutions. ‘CS reward’ means ‘reward in Correct Solutions’, while ‘IS’ means ‘Incorrect Solutions’.

Model	AIME 24	AIME 25	AMC 23	MATH-500	Avg.	Step
Base Model	28.80	22.50	62.90	82.80	49.25	-
TP-GRPO						
- Qwen3-32B	33.12	25.63	64.01	83.81	51.64	140
- Qwen3-4B	33.33	24.58	63.78	83.62	51.33	155
- Gemma-3-12b-it	32.71	23.33	64.76	83.73	51.13	125

Table 5: Robustness Analysis with Different LLMs as PRMs.

4.3 Analysis

4.3.1 Ablation Study

We design two distinct process evaluation mechanisms for correct and incorrect solutions. To better understand their individual contributions, we conduct ablation studies under two settings: one using only process rewards from correct solutions, and the other using only those from incorrect solutions. The corresponding results are presented in Table 4. First, both variants use fewer training steps compared to the full TP-GRPO setup. The reason is that more solutions are filtered out. The results in Table 4 show that both process rewards yield measurable performance gains. We also observe an interesting phenomenon in our experimental results: on AIME 24, using only correct-solution rewards yields better performance, whereas on AIME 25, the opposite holds. We attribute this to the model’s differential problem-solving capability. The base model performs better on AIME 24, enabling more effective learning from correct solutions. In contrast, on AIME 25—where the base model’s accuracy is only 22.50%, making incorrect solutions-driven learning more valuable. Our results show that the proposed process reward mechanism supports effective learning from both correct and incorrect reasoning paths.

4.3.2 Different LRMs for Process Evaluator

To verify that our evaluation scheme imposes low reasoning requirements on the PRM, we conduct an analysis study using different LLMs as the generative PRM. In our main experiments, we adopt Qwen3-32B as the default GenPRM. For analysis, we additionally select two weaker LLMs—Qwen3-4B and Gemma-3-12B-IT [Team et al., 2025a]—as alternative GenPRMs. These three models exhibit progressively weaker reasoning abilities, as evidenced by their accuracy on GPQA Diamond [Rein et al., 2024]: 65.6, 55.9, and 40.9, respectively³. We then apply TP-GRPO using each of the three GenPRMs. The results are shown in Table 5. Despite the significant differences in reasoning capabilities among these models, all GenPRMs consistently improve the performance of the 1.5B LRM through TP-GRPO. This indicates that our method is relatively insensitive to the reasoning ability of the GenPRM. Furthermore, the comparable improvements achieved across different model scales (Qwen3-32B vs. 4B) and families (Qwen3 vs. Gemma) further validate the robustness of our proposed process evaluation mechanism.

Model	AIME 24	AIME 25	AMC 23	Training Steps
On-policy GRPO	32.71	24.58	64.53	850
Off-policy GRPO	30.83	24.79	62.05	950
TP-GRPO (25 steps)	33.54	23.96	64.53	207
TP-GRPO (50 steps)	33.12	25.63	64.01	140

Table 6: Analysis study on the impact of off-policy training.

4.3.3 Analysis on Off-policy Training Pipeline

In this work, we propose an off-policy training pipeline to reduce training time to reduce GPU idling. We further compare this framework with the on-policy method in terms of both reasoning

³Data from: <https://www.datalearner.com/ai-models>

performance and training efficiency, as shown in Table 6. First, it is evident that off-policy GRPO underperforms on-policy GRPO in both accuracy and training efficiency, indicating that on-policy training is more advantageous—consistent with conventional training experience. Figure 3 presents a clearer comparison, where on-policy GRPO demonstrates a more pronounced improvement trend over training steps. To further investigate the impact of update frequency in off-policy training, we reduce the update interval of TP-GRPO’s off-policy pipeline from 50 steps to 25 steps (and to 1 step in an extreme case, which essentially corresponds to true on-policy updates). We observe that both AIME 2024 and AMC 2023 tasks show further performance improvements with shorter update intervals. In summary, although off-policy training is known to be less effective than on-policy training, TP-GRPO—trained under the off-policy setting—still outperforms the on-policy GRPO baseline. This further validates the effectiveness of our proposed GenPRM mechanism.

5 Related Works

5.1 Process Reward Models

PRMs evaluate reasoning trajectories by assigning rewards to each evaluation unit—typically a reasoning step—complementing traditional outcome rewards [Li and Li, 2024, Setlur et al., 2024, Zhang et al., 2025, Younsi et al., 2025]. Depending on the evaluation paradigm, PRMs are broadly categorized into discriminative and generative approaches [Zhao et al., 2025, Zhong et al., 2025]. Discriminative PRMs predict step-level correctness scores via supervised learning [o1 Open Series, 2024b,a], often relying on large-scale human annotations such as in PRM800K [Lightman et al., 2023]. Recent works explore automatic labeling through self-training or simulation [Wang et al., 2023b, Luo et al., 2024, Wang et al., 2024c, Lu et al., 2024b,a]. However, these models often suffer from limited generalization, reward hacking, and poor interpretability [Guo et al., 2025]. In contrast, Generative PRMs frame evaluation as a generation task [Ankner et al., 2024, Zhao et al., 2025], assessing step correctness via generation and using the likelihood of indicative tokens (e.g., “correct”, “yes”) as rewards [Zhang et al., 2024b, Mahan et al., 2024]. Leveraging LLMs’ inherent reasoning abilities, these methods offer better interpretability and scalability [Wang et al., 2023a]. Many adopt an LLM-as-a-Judge framework [Zheng et al., 2023, Gu et al., 2024], prompting pre-trained models without further fine-tuning [Zheng et al., 2024b], while others enhance performance via additional training [Wang et al., 2024b, Khalifa et al., 2025]. Despite progress, key challenges remain, including ambiguous step segmentation and excessive reliance on the PRM’s reasoning capabilities [Guo et al., 2025, Feng et al., 2025].

5.2 PRMs for Optimizing Large Reasoning Models

Prior to the release of DeepSeek R1 [Guo et al., 2025], process rewards have been widely studied as a means to enhance the reasoning capabilities of LLMs [Zhang et al., 2024a, Wang et al., 2024a]. Depending on their application stage, process rewards are used either to guide trajectory pruning during inference [Uesato et al., 2022, Kalra and Tang, 2025, Muennighoff et al., 2025] or to shape training objectives [Zhang et al., 2024c]. This work focuses on the latter. Early methods such as Rest-MCTS* [Zhang et al., 2024a] train PRMs to rank reasoning trajectories and select high-quality paths for learning. Chen et al. [2024b] estimate step-level rewards via question decomposition but ultimately aggregate them into outcome rewards, limiting the effectiveness of fine-grained supervision. Similarly, Li et al. [2024b] also fail to fully exploit process-level signals in their training pipeline. While OpenR [Wang et al., 2024a] employs PRMs to guide policy iteration in LLM training, it remains restricted to the discriminative paradigm. To unlock the full potential of PRMs, we propose a novel evaluation mechanism, including a comprehension-driven evaluation strategy, a thought-level assessment granularity, and a capacity-adaptive reward mechanism.

6 Conclusion and Future Work

In this work, we introduce TP-GRPO, a novel RL framework guided by generative process rewards, with the goal of enhancing training efficiency in math reasoning tasks. TP-GRPO tackles core limitations of conventional process reward approaches by proposing an intrinsic signal-driven, thought-level evaluation mechanism alongside a capacity-adaptive reward formulation that dynamically balances exploration and exploitation. Empirical results show that TP-GRPO achieves superior

improvements compared to the outcome-only RLVR baseline under the same training sample budget, indicating that our designed process evaluation mechanism effectively enhances the optimization efficiency of LRMs.

However, there are inherent risks in overemphasizing process-level rewards. Focusing solely on exploitation while neglecting exploration can lead LRMs to overfit prematurely, thereby impairing their generalization capabilities. Moreover, our experiments do not demonstrate RL scaling through extensive training for TP-GRPO. Nonetheless, as a study centered on the exploration–exploitation trade-off in RL, we argue that a promising direction lies in a dynamic strategy that alternates between process and outcome rewards. Such an approach can simultaneously enhance exploration and exploitation. This is analogous to how students expand the boundaries of their knowledge and abilities by exercising new problems, then consolidate their knowledge through reflection and review. Through this iterative cycle, consistent and efficient improvement can be achieved.

References

- Zachary Ankner, Mansheej Paul, Brandon Cui, Jonathan D Chang, and Prithviraj Ammanabrolu. Critique-out-loud reward models. *arXiv preprint arXiv:2408.11791*, 2024.
- Oded Berger-Tal, Jonathan Nathan, Ehud Meron, and David Saltz. The exploration-exploitation dilemma: a multidisciplinary framework. *PloS one*, 9(4):e95693, 2014.
- Bin Chen, Xinzge Gao, Chuanrui Hu, Penghang Yu, Hua Zhang, and Bing-Kun Bao. Reasongrm: Enhancing generative reward models through large reasoning models. *arXiv preprint arXiv:2506.16712*, 2025a.
- Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. Step-level value preference optimization for mathematical reasoning. *arXiv preprint arXiv:2406.10858*, 2024a.
- Xiuxi Chen, Gaotang Li, Ziqi Wang, Bowen Jin, Cheng Qian, Yu Wang, Hongru Wang, Yu Zhang, Denghui Zhang, Tong Zhang, et al. Rm-r1: Reward modeling as reasoning. *arXiv preprint arXiv:2505.02387*, 2025b.
- Zhaorun Chen, Zhuokai Zhao, Zhihong Zhu, Ruiqi Zhang, Xiang Li, Bhiksha Raj, and Huaxiu Yao. Autoprml: Automating procedural supervision for multi-step reasoning via controllable question decomposition. *arXiv preprint arXiv:2402.11452*, 2024b.
- Zhangying Feng, Qianglong Chen, Ning Lu, Yongqian Li, Siqi Cheng, Shuangmu Peng, Duyu Tang, Shengcai Liu, and Zhirui Zhang. Is prm necessary? problem-solving rl implicitly induces prm capability in llms. *arXiv preprint arXiv:2505.11227*, 2025.
- Ronan Fruit. *Exploration-exploitation dilemma in Reinforcement Learning under various form of prior knowledge*. PhD thesis, Université de Lille 1, Sciences et Technologies; CRISTAL UMR 9189, 2019.
- Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, et al. A survey on llm-as-a-judge. *arXiv preprint arXiv:2411.15594*, 2024.
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.
- Qian Yue Hao, Yiwen Song, Qingmin Liao, Jian Yuan, and Yong Li. Llm-explorer: A plug-in reinforcement learning policy exploration enhancement driven by large language models. *arXiv preprint arXiv:2505.15293*, 2025.
- Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. *arXiv preprint arXiv:2402.14008*, 2024.
- Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.
- Jingcheng Hu, Yinmin Zhang, Qi Han, Daxin Jiang, Xiangyu Zhang, and Heung-Yeung Shum. Open-reasoner-zero: An open source approach to scaling up reinforcement learning on the base model. *arXiv preprint arXiv:2503.24290*, 2025.

- Hugging Face. Open r1: A fully open reproduction of deepseek-r1, January 2025. URL <https://github.com/huggingface/open-r1>.
- Nimit Kalra and Leonard Tang. Verdict: A library for scaling judge-time compute. *arXiv preprint arXiv:2502.18018*, 2025.
- Muhammad Khalifa, Rishabh Agarwal, Lajanugen Logeswaran, Jaekyeom Kim, Hao Peng, Moon-tae Lee, Honglak Lee, and Lu Wang. Process reward models that think. *arXiv preprint arXiv:2504.16828*, 2025.
- Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Huang, Kashif Rasul, Longhui Yu, Albert Q Jiang, Ziju Shen, et al. Numinamath: The largest public dataset in ai4maths with 860k pairs of competition math problems and solutions. *Hugging Face repository*, 13(9):9, 2024a.
- Jiawei Li, Xinyue Liang, Junlong Zhang, Yizhe Yang, Chong Feng, and Yang Gao. Pspo*: An effective process-supervised policy optimization for reasoning alignment. *arXiv preprint arXiv:2411.11681*, 2024b.
- Wendi Li and Yixuan Li. Process reward model with q-value rankings. *arXiv preprint arXiv:2410.11287*, 2024.
- Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let’s verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023.
- Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025a.
- Zijun Liu, Peiyi Wang, Runxin Xu, Shirong Ma, Chong Ruan, Peng Li, Yang Liu, and Yu Wu. Inference-time scaling for generalist reward modeling. *arXiv preprint arXiv:2504.02495*, 2025b.
- Jianqiao Lu, Zhiyang Dou, Hongru Wang, Zeyu Cao, Jianbo Dai, Yunlong Feng, and Zhijiang Guo. Autopsv: Automated process-supervised verifier. *Advances in Neural Information Processing Systems*, 37:79935–79962, 2024a.
- Jianqiao Lu, Zhiyang Dou, Hongru Wang, Zeyu Cao, Jianbo Dai, Yingjia Wan, Yinya Huang, and Zhijiang Guo. Autocv: Empowering reasoning with automated process labeling via confidence variation. *arXiv preprint arXiv:2405.16802*, 2024b.
- Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Meiqi Guo, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, et al. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592*, 2024.
- Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Y. Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Li Erran Li, Raluca Ada Popa, and Ion Stoica. Deepscaler: Surpassing o1-preview with a 1.5b model by scaling rl. <https://pretty-radio-b75.notion.site/DeepScaleR-Surpassing-01-Preview-with-a-1-5B-Model-by-Scaling-RL-19681902c1468005bed8ca303013a4e2>, 2025. Notion Blog.
- Dakota Mahan, Duy Van Phung, Rafael Rafailov, Chase Blagden, Nathan Lile, Louis Castriato, Jan-Philipp Fränken, Chelsea Finn, and Alon Albalak. Generative reward models. *arXiv preprint arXiv:2410.12832*, 2024.
- math ai. Aime 2025 dataset, 2025. URL <https://huggingface.co/datasets/math-ai/aime25>.
- Maxwell-Jia. Aime 2024 dataset, 2024. URL https://huggingface.co/datasets/Maxwell-Jia/AIME_2024.
- Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*, 2025.
- Skywork o1 Open Series. An implementation of generative prm, 2024a. URL <https://github.com/RLHFlow/RLHF-Reward-Modeling>.
- Skywork o1 Open Series. Skywork o1 team, 2024b. URL <https://huggingface.co/Skywork>.

- David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R Bowman. Gpqa: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024.
- Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. Rewarding progress: Scaling automated process verifiers for llm reasoning. *arXiv preprint arXiv:2410.08146*, 2024.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.
- Shuaijie She, Junxiao Liu, Yifeng Liu, Jiajun Chen, Xin Huang, and Shujian Huang. R-prm: Reasoning-driven process reward modeling. *arXiv preprint arXiv:2503.21295*, 2025.
- Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 1279–1297, 2025.
- Mingyang Song, Zhaochen Su, Xiaoye Qu, Jiawei Zhou, and Yu Cheng. Prmbench: A fine-grained and challenging benchmark for process-level reward models. *arXiv preprint arXiv:2501.03124*, 2025.
- Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025a.
- Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*, 2025b.
- 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.
- Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. Trl: Transformer reinforcement learning. <https://github.com/huggingface/trl>, 2020.
- Haoran Wang, Thaleia Zariphopoulou, and Xunyu Zhou. Exploration versus exploitation in reinforcement learning: A stochastic control approach. *arXiv preprint arXiv:1812.01552*, 2018.
- Jiaan Wang, Yunlong Liang, Fandong Meng, Zengkui Sun, Haoxiang Shi, Zhixu Li, Jinan Xu, Jianfeng Qu, and Jie Zhou. Is chatgpt a good nlg evaluator? a preliminary study. *arXiv preprint arXiv:2303.04048*, 2023a.
- Jun Wang, Meng Fang, Ziyu Wan, Muning Wen, Jiachen Zhu, Anjie Liu, Ziqin Gong, Yan Song, Lei Chen, Lionel M Ni, et al. Openr: An open source framework for advanced reasoning with large language models. *arXiv preprint arXiv:2410.09671*, 2024a.
- Peiyi Wang, Lei Li, Zhihong Shao, RX Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. *arXiv preprint arXiv:2312.08935*, 2023b.
- Tianlu Wang, Ilia Kulikov, Olga Golovneva, Ping Yu, Weizhe Yuan, Jane Dwivedi-Yu, Richard Yuanzhe Pang, Maryam Fazel-Zarandi, Jason Weston, and Xian Li. Self-taught evaluators. *arXiv preprint arXiv:2408.02666*, 2024b.
- Yue Wang, Qiuzhi Liu, Jiahao Xu, Tian Liang, Xingyu Chen, Zhiwei He, Linfeng Song, Dian Yu, Juntao Li, Zhuosheng Zhang, et al. Thoughts are all over the place: On the underthinking of o1-like llms. *arXiv preprint arXiv:2501.18585*, 2025.
- Zihan Wang, Yunxuan Li, Yuexin Wu, Liangchen Luo, Le Hou, Hongkun Yu, and Jingbo Shang. Multi-step problem solving through a verifier: An empirical analysis on model-induced process supervision. *arXiv preprint arXiv:2402.02658*, 2024c.
- Yifei Xu, Tusher Chakraborty, Srinagesh Sharma, Leonardo Nunes, Emre Kıcıman, Songwu Lu, and Ranveer Chandra. Direct reasoning optimization: Llms can reward and refine their own reasoning for open-ended tasks. *arXiv preprint arXiv:2506.13351*, 2025.

- Zihuiwen Ye, Luckeciano Carvalho Melo, Younesse Kaddar, Phil Blunsom, Sam Staton, and Yarin Gal. Uncertainty-aware step-wise verification with generative reward models. *arXiv preprint arXiv:2502.11250*, 2025.
- Adam Younsi, Abdalgader Abubaker, Mohamed El Amine Seddik, Hakim Hacid, and Salem Lahlou. Accurate and diverse llm mathematical reasoning via automated prm-guided gflownets. *arXiv preprint arXiv:2504.19981*, 2025.
- Qiyang Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.
- Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *arXiv preprint arXiv:2504.13837*, 2025.
- Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. Rest-mcts*: Llm self-training via process reward guided tree search. *Advances in Neural Information Processing Systems*, 37:64735–64772, 2024a.
- 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*, 2024b.
- Yuxiang Zhang, Shangxi Wu, Yuqi Yang, Jiangming Shu, Jinlin Xiao, Chao Kong, and Jitao Sang. o1-coder: an o1 replication for coding. *arXiv preprint arXiv:2412.00154*, 2024c.
- Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. The lessons of developing process reward models in mathematical reasoning. *arXiv preprint arXiv:2501.07301*, 2025.
- Jian Zhao, Runze Liu, Kaiyan Zhang, Zhimu Zhou, Junqi Gao, Dong Li, Jiafei Lyu, Zhouyi Qian, Biqing Qi, Xiu Li, et al. Genprm: Scaling test-time compute of process reward models via generative reasoning. *arXiv preprint arXiv:2504.00891*, 2025.
- Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. ProcessBench: identifying process errors in mathematical reasoning. *arXiv preprint arXiv:2412.06559*, 2024a.
- Chujie Zheng, Zhenru Zhang, Beichen Zhang, Runji Lin, Keming Lu, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. Processbench: Identifying process errors in mathematical reasoning. *arXiv preprint arXiv:2412.06559*, 2024b.
- Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.
- Jialun Zhong, Wei Shen, Yanzeng Li, Songyang Gao, Hua Lu, Yicheng Chen, Yang Zhang, Wei Zhou, Jinjie Gu, and Lei Zou. A comprehensive survey of reward models: Taxonomy, applications, challenges, and future. *arXiv preprint arXiv:2504.12328*, 2025.
- Jiaru Zou, Ling Yang, Jingwen Gu, Jiahao Qiu, Ke Shen, Jingrui He, and Mengdi Wang. Reasonflux-prm: Trajectory-aware prms for long chain-of-thought reasoning in llms. *arXiv preprint arXiv:2506.18896*, 2025.

A Proof of Proposition

Proposition 1. For a correct solution o_i , token $o_{i,t}$ in correct thoughts have advantage $A_{i,t} = \hat{r}_i^o$; while token $o_{i,t}$ in incorrect thoughts have advantage $A_{i,t} = \hat{r}_i^o - r^{\text{correct}}$.

Proof. The proof is straightforward. After merging steps of the same type, the original *think* can be decomposed into an alternating sequence of correct and incorrect thoughts:

$$\text{think} = \text{thought}_1^{\text{correct}} \circ \text{thought}_2^{\text{incorrect}} \circ \dots \circ \text{thought}_{n-1}^{\text{incorrect}} \circ \text{thought}_n^{\text{correct}}.$$

Accordingly, the thought-level reward sequence is:

$$[r_{\text{index}(1)}, r_{\text{index}(2)}, \dots, r_{\text{index}(n)}] = [r^{\text{correct}}, -r^{\text{correct}}, \dots, -r^{\text{correct}}, \hat{r}_i^o],$$

where $\text{index}(j)$ denotes the index of the last token in the j -th thought. The last thought t_n^{correct} receives only the outcome reward \hat{r}^o .

For tokens in a correct thought $\text{thought}_j^{\text{correct}}$, the advantage value is computed as:

$$A_{\text{index}(j-1) < k \leq \text{index}(j)}^{\text{correct}} = r_{\text{index}(j)} + r_{\text{index}(j+1)} + \dots + r_{\text{index}(n)}. \quad (8)$$

Given the alternating reward pattern, this simplifies to:

$$\underbrace{r^{\text{correct}} + \dots + r^{\text{correct}}}_{(n-j)/2 \text{ terms}} + \underbrace{(-r^{\text{correct}}) + \dots + (-r^{\text{correct}})}_{(n-j)/2 \text{ terms}} + \hat{r}^o = \hat{r}^o.$$

Similarly, for tokens in the incorrect thought $\text{thought}_i^{\text{incorrect}}$, the corresponding advantage is:

$$\begin{aligned} A_{\text{index}(j-1) < k \leq \text{index}(j)}^{\text{correct}} &= r_{\text{index}(j)} + r_{\text{index}(j+1)} + \dots + r_{\text{index}(n)} \\ &= \underbrace{r^{\text{correct}} + \dots + r^{\text{correct}}}_{(n-j+1)/2 - 1 \text{ times}} + \underbrace{(-r^{\text{correct}} - \dots - r^{\text{correct}})}_{(n-j+1)/2 \text{ times}} + \hat{r}^o \\ &= \hat{r}^o - r^{\text{correct}} \end{aligned} \quad (9)$$

□

Proposition 2. For an incorrect solution o_i , token c_t in matched thoughts have advantage $A_{i,t} = \hat{r}_i^o$, while token c_t in unmatched thoughts have advantage $A_{i,t} = 0$.

Proof. Following the same reasoning as in Proposition 1, for an incorrect *think*, we first merge steps of the same type and decompose the sequence into alternating matched and unmatched thoughts with respect to the *answer*. We assign a penalty to matched thoughts with reward $r^{\text{incorrect}} = \hat{r}^o$, and reward unmatched thoughts with $-r^{\text{incorrect}} = -\hat{r}^o$.

For a matched thought t_j^{matched} , the advantage of its tokens is computed as:

$$\begin{aligned} A_{\text{index}(j-1) < k \leq \text{index}(j)}^{\text{matched}} &= r_{\text{index}(j)} + r_{\text{index}(j+1)} + \dots + r_{\text{index}(n)} \\ &= \underbrace{\hat{r}^o + \dots + \hat{r}^o}_{(n-j)/2 \text{ terms}} + \underbrace{(-\hat{r}^o) + \dots + (-\hat{r}^o)}_{(n-j)/2 \text{ terms}} + \hat{r}^o \\ &= \hat{r}^o. \end{aligned} \quad (10)$$

Similarly, for an unmatched thought $t_j^{\text{unmatched}}$, the advantage is:

$$\begin{aligned} A_{\text{index}(j-1) < k \leq \text{index}(j)}^{\text{unmatched}} &= r_{\text{index}(j)} + r_{\text{index}(j+1)} + \dots + r_{\text{index}(n)} \\ &= \underbrace{\hat{r}^o + \dots + \hat{r}^o}_{(n-j+1)/2 - 1 \text{ terms}} + \underbrace{(-\hat{r}^o) + \dots + (-\hat{r}^o)}_{(n-j+1)/2 \text{ terms}} + \hat{r}^o \\ &= 0. \end{aligned} \quad (11)$$

□

B More Experiment Settings

B.1 Training Settings

During process evaluation, we employ Qwen3-32B as the default evaluator. Due to the high computational cost of process evaluation, we implement some heuristic rules to skip some solutions that are unlikely to obtain process rewards:

- For solutions with correct answers, if the *think* is short (fewer than 4,096 characters), we skip directly, as such *think* typically contains only correct steps.

Hyperparameter	Value
<i>Sampling Configuration</i>	
system prompt	{problem} Let’s think step by step and output the final answer within \boxed{ }.
temperature	1.0
max prompt length	1024
max response length	8192
rollout size	8
<i>Process Evaluation Configuration</i>	
default model	Qwen3-32B
backend	VLLM
temperature	0.7
max tokens	8192
tensor-parallel-size(VLLM)	2
max_num_seqs(VLLM)	512
<i>Training Configuration</i>	
gradient accumulation steps	1
learning rate	1.0e-06
lr scheduler type	constant
max steps	50
max iteration num	20
per device train batch size	5
process reward coefficient α	1
ϵ_{low}	0.2
ϵ_{high}	0.28
<i>Evaluation Configuration</i>	
max tokens	32768
temperature	0.6
top P	0.95
top K	-1
gpu_memory_utilization	1.0

Table 7: All key configuration hyperparameters used in this paper.

- For solutions with incorrect answers, if the *answer* is very short (fewer than 256 characters), we skip directly, as these *answer* rarely contains obvious reasoning steps.
- Solutions that are truncated due to length constraints are also skipped.

In the implementation of TP-GRPO and all replicated baseline experiments, we select GRPO as the base algorithm, and incorporate two modifications proposed in DAPO: ϵ_{high} and token-level loss. The outcome reward function combines accuracy and format rewards. We extract the answer from ‘\boxed{ }’ and compare it against the golden answer: a perfect match yields accuracy reward=1, otherwise 0. Additionally, we observed collapse when reproducing the on-policy GRPO baseline with $\beta = 0$. Therefore, following the open-r1 implementation [Hugging Face, 2025], we set $\beta = 0.04$ for the GRPO baseline. Additional learning-related hyperparameters and configurations are summarized in Table 7.

B.2 Evaluation Settings

All evaluation experiments—including those for TP-GRPO and all baselines—are conducted using the evaluation codes from the **rllm** project⁴.

⁴rllm: <https://github.com/agentica-project/rllm>

C More Experiments Results

C.1 Case Studies

To intuitively compare the improvements of our proposed TP-GRPO algorithm over the original GRPO, we selected one problem each from the AIME24 and AIME25. These problems were then presented to the DeepSeek-R1-Distill-Qwen-1.5B and DeepSeek-R1-Distill-Qwen-7B models, which were trained using both algorithms respectively, to observe their corresponding reasoning processes. The detailed results for the DeepSeek-R1-Distill-Qwen-1.5B model are shown in Table 8 and Table 9. As observed from the model’s thinking content, the model post-trained with the TP-GRPO algorithm can correct the specific reasoning errors made by the model trained with the original GRPO. This provides an intuitive demonstration of the effectiveness of our proposed algorithm.

1.5B Model with GRPO Case Study

[Question]

Alice chooses a set A of positive integers. Then Bob lists all finite nonempty sets B of positive integers with the property that the maximum element of B belongs to A . Bob’s list has 2024 sets. Find the sum of the elements of A . Let’s think step by step and output the final answer within `\boxed{}`.

[Model Thinking Content]

Okay, so Alice picks a set A of positive integers. Then Bob lists all finite nonempty sets B of positive integers where the maximum element of B is in A . The total number of such sets B is 2024. We need to find the sum of the elements of A .

...

So, if A has n elements, then the total number of such sets B is the sum from $k=1$ to n of 2^{k-1} . Because for each element a_k in A , the number of subsets where a_k is the maximum is 2^{k-1} . Therefore, the total number of sets B is $2^n - 1$, since the sum from $k=0$ to $n-1$ of 2^k is $2^n - 1$. So, if A has elements a_1, a_2, \dots, a_n , then the total number is $2^{a_1-1} + 2^{a_2-1} + \dots + 2^{a_n-1} = 2024$

...

Now, 2024 can be expressed as a sum of powers of 2. Let’s see, 2024 in binary is 1111101100, which is $1024 + 512 + 256 + 128 + 64 + 32 + 4$. Wait, but let me compute that.

Wait, 1024 is 2^{10} , 512 is 2^9 , 256 is 2^8 , 128 is 2^7 , 64 is 2^6 , 32 is 2^5 , 16 is 2^4 , 8 is 2^3 , 4 is 2^2 , 2 is 2^1 , 1 is 2^0 . So, $1024 + 512$ is 1536, plus 256 is 1792, plus 128 is 1920, plus 64 is 1984, plus 32 is 2016, plus 4 is 2020. Wait, that’s $1024 + 512 + 256 + 128 + 64 + 32 + 4 = 2020$. Hmm, but 2024 is 4 more, so $2020 + 4$ is 2024. So, in terms of powers of 2, $2024 = 2^{10} + 2^9 + 2^8 + 2^7 + 2^6 + 2^5 + 2^2$.

...

So, the exponents $a_i - 1$ are 2, 5, 6, 7, 8, 9, 10. Therefore, a_i are 3, 6, 7, 8, 9, 10, 11. Therefore, the set A is 3, 6, 7, 8, 9, 10, 11. Therefore, the sum of the elements of A is $3 + 6 + 7 + 8 + 9 + 10 + 11$.

...

Therefore, the sum of the elements of A is $3 + 6 + 7 + 8 + 9 + 10 + 11$.

Let me compute that:

$3 + 6$ is 9.

$9 + 7$ is 16.

$16 + 8$ is 24.

$24 + 9$ is 33.

$33 + 10$ is 43.

$43 + 11$ is 54.

So, the sum is 54.

Therefore, the answer is 54.

Final Answer: The sum of the elements of A is $\boxed{54}$.

Table 8: Case study of DeepSeek-R1-Distill-Qwen-1.5B with GRPO. For the given problem, the model incorrectly answered 54, while the correct answer is 55. Due to the model’s extensive thinking process, we have excerpted the key steps for clarity. As shown, the error (highlighted by red) stems from a miscalculation in the binary representation of 2024.

1.5B Model with TP-GRPO Case Study

[Question]

Alice chooses a set A of positive integers. Then Bob lists all finite nonempty sets B of positive integers with the property that the maximum element of B belongs to A . Bob’s list has 2024 sets. Find the sum of the elements of A . Let’s think step by step and output the final answer within `\boxed{}`.

[Model Thinking Content]

Okay, so I have this problem here where Alice chooses a set A of positive integers. Then Bob lists all finite nonempty sets B of positive integers with the property that the maximum element of B belongs to A . Bob’s list has 2024 sets. I need to find the sum of the elements of A . Hmm, let me try to break this down step by step.

...

So for each a in A , the number of such sets is equal to the number of subsets of $1, 2, \dots, a-1$. Because each subset can be combined with a to form a set B with maximum a .

Therefore, for each a in A , the number of sets B with maximum a is 2^{a-1} . Because the number of subsets of a set with n elements is 2^n . So in this case, n is $a-1$, so it’s 2^{a-1} .

So, if A is a set of numbers, say $A = a_1, a_2, \dots, a_k$, then the total number of sets B that Bob can list is the sum over each a_i in A of 2^{a_i-1} . Because each a_i contributes 2^{a_i-1} sets. Therefore, the total number of sets B is the sum of 2^{a_i-1} for all a_i in A . So, let me denote that $S = \sum_a 2^{a-1} = 2024$.

...

So, let’s factor 2024 into its prime factors to see if we can represent it as a sum of powers of 2.

First, 2024 divided by 2 is 1012, divided by 2 again is 506, divided by 2 again is 253. 253 is not divisible by 2, so the prime factors are $2^3 \times 253$.

Now, 253 divided by 11 is 23, because 11×23 is 253. So, $2024 = 2^3 \times 11 \times 23$. So, $2024 = 8 \times 11 \times 23$. Hmm, interesting.

But how does this help me? I need to express 2024 as the sum of 2^{a-1} terms, each of which is a power of 2, and each a is a positive integer. So, each term is a power of 2, and the sum is 2024. So, to find A , I need to express 2024 as the sum of distinct powers of 2

...

So, let me compute the binary representation of 2024. Calculating:

$$2^{10} = 1024$$

$$2^9 = 512$$

$$2^8 = 256$$

$$2^7 = 128$$

$$2^6 = 64$$

$$2^5 = 32$$

$$2^3 = 8$$

So, summing these up:

$$1024 + 512 = 1536$$

$$1536 + 256 = 1792$$

$$1792 + 128 = 1920$$

$$1920 + 64 = 1984$$

$$1984 + 32 = 2016$$

$$2016 + 8 = 2024$$

Yes, that adds up correctly. So, 2024 in binary is 1111101000. So, the binary representation has 1s at positions 10,9,8,7,6,5,3. So, that would mean that the exponents in the sum are 10,9,8,7,6,5,3.

...

So, if the exponents are 10,9,8,7,6,5,3, then the corresponding a's are 11,10,9,8,7,6,4.
Therefore, set A is 4,6,7,8,9,10,11.

So, the sum of elements in A is $4 + 6 + 7 + 8 + 9 + 10 + 11$.

Let me compute that:

$$4 + 6 = 10$$

$$10 + 7 = 17$$

$$17 + 8 = 25$$

$$25 + 9 = 34$$

$$34 + 10 = 44$$

$$44 + 11 = 55$$

So, the sum is 55.

...

Final Answer: The sum of the elements of A is 55.

Table 9: Case study of DeepSeek-R1-Distill-Qwen-1.5B with TP-GRPO. Using the same problem from Table 8, we also present only the key steps. Post-training with TP-GRPO, the model successfully computes the binary representation of 2024 and yields the correct final answer. We use blue to highlight these steps.

7B Model with GRPO Case Study

[Question]

Let A be the set of positive integer divisors of 2025. Let B be a randomly selected subset of A . The probability that B is a nonempty set with the property that the least common multiple of its elements is 2025 is $\frac{m}{n}$, where m and n are relatively prime positive integers. Find $m + n$. Let's think step by step and output the final answer within `\boxed{}`.

[Model Thinking Content]

Okay, so I have this problem here where I need to find the probability that a randomly selected subset B of the set A (which consists of all positive integer divisors of 2025) has the property that the least common multiple (LCM) of its elements is 2025.

...

I know that 2025 divided by 25 is 81, because 25 times 81 is 2025. And 81 is 9 squared, so 2025 is 25 times 81, which is 5 squared times 3 to the fourth power. So, $2025 = 5^2 3^4$. Therefore, the prime factorization of 2025 is $5^2 3^4$.

Now, to find all the positive divisors of 2025, I can use the exponents of its prime factors. The number of divisors is given by multiplying one more than each of the exponents in the prime factorization. So here, for 5^2 , the exponent is 2, so we have $2 + 1 = 3$. For 3^4 , the exponent is 4, so $4 + 1 = 5$. Therefore, the total number of divisors is $3 \times 5 = 15$. So, set A has 15 elements.

...

Next, we need to consider all possible subsets B of A , and find the probability that the LCM of the elements in B is 2025. Since we're dealing with subsets, the total number of subsets is 2^{15} , because each element can either be included or excluded. However, the problem specifies that B is a nonempty set, so we have to subtract 1 for the empty set. **Therefore, the total number of possible nonempty subsets is $2^{15} - 1$.**

Wait, actually, let me think. The problem says "a randomly selected subset of A ." It doesn't specify whether it's nonempty or not. But then, it goes on to specify that B is a nonempty set with the property that LCM of its elements is 2025. So, perhaps the probability is considering all subsets, including the empty set, but the event we're interested in is only when B is nonempty. Hmm, the wording is a bit ambiguous.

Looking back at the problem: "The probability that B is a nonempty set with the property that the least common multiple of its elements is 2025." So, it's given that B is a nonempty

set, so we can consider the total number of subsets as $2^{15} - 1$, since we exclude the empty set. Therefore, the probability is $\frac{(\text{number of desired subsets})}{(2^{15}-1)}$.

...

Given that $2025 = 3^4 5^2$, the LCM of a subset B is 2025 if and only if the subset contains elements that cover all the prime factors with their maximum exponents.

...

By inclusion-exclusion, this number is $U - |A| - |B| + |A \cap B|$.

So, $|A|$ is the number of subsets without 81. Since 81 is one specific element, the number of subsets without 81 is 2^{14} .

Similarly, $|B|$ is the number of subsets without 25, which is also 2^{14} .

$|A \cap B|$ is the number of subsets without both 81 and 25, which is 2^{13} .

Therefore, the number of subsets containing at least one 81 and at least one 25 is:

$$2^{15} - 2^{14} - 2^{14} + 2^{13}.$$

Let me compute this:

$$2^{15} = 32768$$

$$2^{14} = 16384$$

$$2^{13} = 8192$$

$$\text{So, } 32768 - 16384 - 16384 + 8192.$$

Compute step by step:

$$32768 - 16384 = 16384$$

$$16384 - 16384 = 0$$

$$0 + 8192 = 8192$$

So, the number of subsets containing at least one 81 and at least one 25 is 8192.

...

Therefore, the probability is $\frac{8191}{(2^{15}-1)} = \frac{8191}{32767}$.

Now, we need to simplify this fraction. Let's see if 8191 and 32767 have any common factors.

First, let me factorize 32767. I know that $2^{15} - 1$, which is $32768 - 1 = 32767$.

32767 is a Mersenne number, $2^{15} - 1$. Let's see if it's prime.

Well, 2^{15} is 32768, so 32767 is $32768 - 1$. Let me check its divisibility.

$32767 : 74681 = 32767$? Let's check: $7 \times 4000 = 28000$, $7 \times 681 = 4767$, so $28000 + 4767 = 32767$. Yes, so $32767 = 7 \times 4681$.

Now, let's factor 4681. Let me check if 4681 is divisible by 7: $7 \times 668 = 4676$, $4681 - 4676 = 5$, so no. Next, 11: $4 - 6 + 8 - 1 = 5$, which is not divisible by 11. 13: $4681 \div 13$. $13 \times 360 = 4680$, so $4681 - 4680 = 1$, so no. 17: $17 \times 275 = 4675$, $4681 - 4675 = 6$, not divisible. 19: $19 \times 246 = 4674$, $4681 - 4674 = 7$, no. 23: $23 \times 203 = 4669$, $4681 - 4669 = 12$, no. 29: $29 \times 161 = 4669$, same as above. 31: $31 \times 151 = 4681$? Let's compute $31 \times 150 = 4650$, plus 31 = 4681. Yes! So, $4681 = 31 \times 151$.

Therefore, $32767 = 7 \times 31 \times 151$.

Now, let's check if 8191 is divisible by any of these primes: 7, 31, 151.

$8191 \div 7$: $7 \times 1170 = 8190$, so $8191 - 8190 = 1$, so remainder 1. Not divisible by 7.

$8191 \div 31$: Let's see, $31 \times 264 = 8184$, $8191 - 8184 = 7$, so remainder 7. Not divisible by 31.

$8191 \div 151$: $151 \times 54 = 8154$, $8191 - 8154 = 37$, which is less than 151, so not divisible.

Therefore, 8191 and 32767 share no common factors besides 1, so the fraction $\frac{8191}{32767}$ is already in its simplest form.

...

Therefore, $m = 8191$, $n = 32767$, and $m + n = 40958$.

I think that's solid. I don't see any mistakes in the reasoning.

Final Answer: 40958

Table 10: Case study of DeepSeek-R1-Distill-Qwen-7B with GRPO. For the given problem, the model incorrectly answered 40958, while the correct answer is 237. Due to the model's extensive thinking process, we have excerpted the key steps for clarity. As shown, the error (highlighted by red) mainly from a miscalculation in the number of corresponding subsets.

7B Model with TP-GRPO Case Study

[Question]

Let A be the set of positive integer divisors of 2025. Let B be a randomly selected subset of A . The probability that B is a nonempty set with the property that the least common multiple of its elements is 2025 is $\frac{m}{n}$, where m and n are relatively prime positive integers. Find $m + n$. Let's think step by step and output the final answer within `\boxed{\}`.

[Model Thinking Content]

Okay, so I have this problem here where I need to find the probability that a randomly selected subset B of the set A (which consists of all positive integer divisors of 2025) has the property that the least common multiple (LCM) of its elements is 2025.

...
 A is the set of all positive integer divisors of 2025. So, let me factorize 2025 to find out its divisors. 2025 is a number I might recognize as a square. Let me check: 45×45 is 2025. So, $2025 = 45^2$. But 45 itself can be broken down further: $45 = 9 \times 5$, and 9 is 3^2 . So, 2025 is $3^4 \times 5^2$. Therefore, the prime factorization of 2025 is $3^4 \times 5^2$.

Now, to find the number of positive divisors, we can use the formula for the number of divisors based on prime factorization. If a number N has the prime factorization $N = p_1^{a_1} * p_2^{a_2} * \dots * p_n^{a_n}$, then the number of positive divisors is $(a_1 + 1)(a_2 + 1) \dots (a_n + 1)$. So, for 2025, which is $3^4 \times 5^2$, the number of divisors is $(4 + 1) \times (2 + 1) = 5 \times 3 = 15$. So, set A has 15 elements.

...
Next, I need to consider the power set of A , which is all possible subsets of A , including the empty set and A itself. The number of subsets of a set with n elements is 2^n . Since A has 15 elements, the power set has 2^{15} subsets. Therefore, the total number of possible subsets B is 32768.

Now, the problem is asking for the probability that a randomly selected subset B has an LCM of 2025. So, I need to find how many such subsets exist where the LCM of their elements is 2025, and then divide that by the total number of subsets, which is 32768.

...
Given that 2025 is $3^4 \times 5^2$, the LCM of a subset will be 2025 if and only if in the subset, there is at least one number with 3^4 and at least one number with 5^2 . Wait, is that correct? Or is it sufficient that the maximum exponent of 3 in the subset is 4 and the maximum exponent of 5 is 2?

Yes, exactly. So, for the LCM to be 2025, the subset must include numbers such that the highest power of 3 in the subset is 3^4 and the highest power of 5 is 5^2 . So, in other words, the subset must contain at least one multiple of 3^4 (which is 81) and at least one multiple of 5^2 (which is 25).

...
Therefore, the number of subsets with LCM 2025 is equal to the number of subsets that include at least one multiple of 81 and at least one multiple of 25. So, to compute this, we can use the principle of inclusion-exclusion.

First, let's denote:

- Let S be the set A , which has 15 elements.
- Let X be the set of subsets of S that include at least one multiple of 81.
- Let Y be the set of subsets of S that include at least one multiple of 25.

We need to find $|X \cap Y|$, which is the number of subsets that include at least one multiple of 81 and at least one multiple of 25.

By the principle of inclusion-exclusion:

$$|X \cap Y| = |X| + |Y| - |X \cup Y|$$

So, let's compute each term.

Total subsets: 2^{15} .

Subsets without any multiple of 81: So, we need to count the number of subsets that do not include 81. 2025 is $3^4 \times 5^2$, so the divisors are of the form $3^a \times 5^b$ where a is 0,1,2,3,4 and b is 0,1,2.

The number of divisors is 15. The number of divisors that are multiples of 81: 81 is 3^4 , so any multiple of 81 must have $a=4$. So, the number of such divisors is equal to the

number of possible b's, which are 0,1,2. So, there are 3 divisors that are multiples of 81: $81 \times 1 = 81$, $81 \times 5 = 405$, $81 \times 25 = 2025$.

Therefore, the number of elements not divisible by 81 is $15 - 3 = 12$. Therefore, the number of subsets without any multiple of 81 is 2^{12} .

Similarly, subsets without any multiple of 25: 25 is 5^2 . So, the divisors that are multiples of 25 are those with $b=2$. The number of such divisors is equal to the number of a's, which are 0,1,2,3,4. So, that's 5 divisors: 1, 5, 25, 75, 225, 2025? Wait, no: 1 is $3^0 \times 5^0$, 5 is $3^0 \times 5^1$, 25 is $3^0 \times 5^2$, 75 is $3^1 \times 5^2$, 225 is $3^2 \times 5^2$, 2025 is $3^4 \times 5^2$. Wait, that's 6 divisors. Wait, but 2025 is $3^4 \times 5^2$, so for $b=2$, a can be 0,1,2,3,4: that's 5 divisors. Wait, $3^0 \times 5^2$, $3^1 \times 5^2$, ..., $3^4 \times 5^2$: that's 5 divisors. Wait, why did I think 6? Hmm, maybe I confused with 2025's total divisors.

Wait, no, the number of divisors where $b=2$ is 5 because a can be 0,1,2,3,4. So, 5 divisors. Therefore, the number of divisors not divisible by 25 is $15 - 5 = 10$. Therefore, the number of subsets without any multiple of 25 is 2^{10} .

Now, the subsets that include neither any multiple of 81 nor any multiple of 25: So, these are subsets that do not contain any of the 3 multiples of 81 and do not contain any of the 5 multiples of 25. So, how many such elements are there?

Total elements: 15.

Number of elements that are multiples of 81 or 25: Let's compute this using inclusion-exclusion. Number of multiples of 81: 3, multiples of 25: 5, and multiples of both 81 and 25: Well, 81 and 25 are coprime, so their least common multiple is $81 \times 25 = 2025$. So, only 2025 is a multiple of both 81 and 25. Therefore, the number of elements that are multiples of 81 or 25 is $3 + 5 - 1 = 7$.

Therefore, the number of elements that are neither multiples of 81 nor multiples of 25 is $15 - 7 = 8$.

Therefore, the number of subsets that include neither any multiple of 81 nor any multiple of 25 is 2^8 .

Therefore, putting it all together:

$|X \cap Y| = \text{Total subsets} - \text{subsets without 81} - \text{subsets without 25} + \text{subsets without both 81 and 25}$

Which is:

$$|X \cap Y| = 2^{15} - 2^{12} - 2^{10} + 2^8$$

Let me compute these values step by step.

$$\text{First, } 2^{15} = 32768$$

$$2^{12} = 4096$$

$$2^{10} = 1024$$

$$2^8 = 256$$

So, substituting:

$$|X \cap Y| = 32768 - 4096 - 1024 + 256$$

Let me compute this step by step:

$$\text{First, } 32768 - 4096 = 28672$$

$$\text{Then, } 28672 - 1024 = 27648$$

$$\text{Then, } 27648 + 256 = 27904$$

$$\text{So, } |X \cap Y| = 27904$$

Therefore, the number of subsets B where $\text{LCM}(B) = 2025$ is 27904.

...

Therefore, the probability is $\frac{27904}{32768}$.

Now, we need to reduce this fraction to its lowest terms. Let's find the greatest common divisor (GCD) of 27904 and 32768.

First, let's note that both numbers are divisible by 16: $27904 \div 16 = 1744$, $32768 \div 16 = 2048$.

So, 1744 and 2048.

Again, both are divisible by 16: $1744 \div 16 = 109$, $2048 \div 16 = 128$.

So, 109 and 128. Now, 109 is a prime number (since it's not divisible by 2,3,5,7,11; $\sqrt{109} \approx 10.4$, so primes up to 11 are tested). 128 is 2^7 . Since 109 is prime and doesn't divide 128, the GCD is 1.

Therefore, the reduced fraction is $\frac{1744}{2048}$. Wait, no: wait, $27904 \div 16 = 1744$, $32768 \div 16 = 2048$. Then, $1744 \div 16 = 109$, $2048 \div 16 = 128$. So, the reduced fraction is $\frac{109}{128}$.
Wait, but $1744 \div 16 = 109$? Wait, $16 * 109 = 1744$? Let me compute $16 * 100 = 1600$, $16 * 9 = 144$, so $1600 + 144 = 1744$. Yes, correct. Similarly, $16 * 128 = 2048$, which is correct.
So, 109 and 128 are coprime, so the reduced fraction is $\frac{109}{128}$.
Therefore, the probability is $\frac{109}{128}$, so $m=109$, $n=128$.
Thus, $m + n = 109 + 128 = 237$.
Wait, hold on, but let me confirm the calculations once more because 27904 divided by 32768 reduces by 16 twice, leading to $\frac{109}{128}$, which is correct.
...

Therefore, the probability is $\frac{27904}{32768} = \frac{109}{128}$, so $m + n = 237$.
Thus, I think that is the correct answer.

Final Answer: 237

Table 11: Case study of DeepSeek-R1-Distill-Qwen-7B with TP-GRPO. Using the same problem from Table 10, we also present only the key steps. Post-training with TP-GRPO, the model successfully computes the number of corresponding subsets and yields the correct final answer. We use blue to highlight these steps.

D Process Evaluation Details

In this part, we will provide the implementation details of the process evaluation with GenPRM.

Atomic Logical Actions-Based Decompose Details In our process, a solution must first be decomposed into a series of steps with appropriate granularity for subsequent processing. However, since solutions generated by the model are often lengthy, it is difficult to directly decompose them into steps that are both content-complete and have clear boundaries. Here, we first split each solution into shorter sentences based on the "\n\n" delimiter. However, we observed in practice that the model often generates "\n\n" in the middle of a process, leading to severe fragmentation of the resulting sentences and a vast number of segments. Therefore, through observation we defined a list of punctuation marks that signify the end of the finest-grained meaningful sentences. Based on this list, we sequentially merge consecutive short sentences until the current sentence terminates with one of these specified punctuation marks. The punctuation marks list is shown in Table 12. To facilitate subsequent processing, we then insert a sequentially increasing label (e.g., «0», «1», ...) at the beginning of each merged sentence. To ensure that each sentence has an appropriate semantic granularity, we define eight atomic logical actions and prompt GenPRM to further merge consecutive sentences based on these actions. Ultimately, for each solution we obtain a series of steps with appropriate quantity and granularity, as well as clear semantic boundaries. The prompt used is shown in Table 14. For incorrect solutions, we also need to decompose their corresponding answers. Since answers are typically much shorter than solutions, we directly decompose it using the atomic logical actions mentioned above, bypassing the preprocessing stage. Specifically, we instruct GenPRM to enclose each delineated step within "<step>" and "</step>" tags. It facilitates the subsequent parsing of steps using regular expressions, and mitigates the risk of GenPRM corrupting the original answer (e.g., by rewriting or omitting parts). The prompt used is shown in Table 19.

Subsequent Match-Based Step Type Check for Incorrect Solutions Details For solutions that remain excessively long (i.e., exceeding 8192 characters), after the initial decomposition we apply an additional step-level summarization using GenPRM. This makes them more concise for subsequent processing. The prompt used is shown in Table 13. Subsequently, to match each step of the answer to the corresponding steps in the solution, we focus solely on semantic and content similarity, disregarding whether the solution steps themselves are correct or not. The prompt used can be found in Table 15.

Subsequent Reflection-Based Step Type Check for Correct Solutions Details For correct solutions, we first provide the step-wise decomposed solution to GenPRM, tasking it with identifying all reflection segments. However, solution may contains validation-style reflections (e.g., "double-checking") that do not identify specific errors. We constrain the model to only find reflection segments

that point out concrete errors. The prompt used is shown in Table 16. Next, we iterate through each of the identified reflection segments to trace its error source, which we define as the earliest step where the error mentioned in the reflection first appeared. The prompt used is shown in Table 17. After identifying the error source for each reflection segment, we retain only the source with the minimum index, as this is crucial for meaningfully identifying the ultimate origin of the error. Specifically, if multiple reflection segments map to the earliest source, we keep the reflection segment with the maximum index. Based on the principle of semantic consistency, all reflections within this span are considered semantically related to the source error, either directly or as an indirect consequence. Finally, we task GenPRM to determine the type of each step within it. Specifically, we define three types: **Incorrect**, **Reflection**, and **Correct**.

- **Incorrect:** An Incorrect step is one that contains a logical or computational error.
- **Reflection:** A Reflection step is one that contains a reflective action.
- **Correct:** Steps that fall into neither of the two aforementioned types.

The prompt used for this classification task is shown in Table 18. As our overall workflow is divided into three stages, the actual implementation involves parsing the JSON data returned from the prompt of the preceding stage, and then passing to the prompt for the subsequent stage.

Punctuation Marks List
[., ...,,,, !, ?]

Table 12: The punctuation marks list we use to merge consecutive pre-split sentences.

E More details on Evaluation Prompts

Summarize Solution Prompt
<p>You will be presented with the following two contents:</p> <p>[Math Problem] {problem}</p> <p>[Solution] {solution in a list format}</p> <p>This thought process includes $len(solution)$ Steps in total. Your task is to summarize the main content of these all Steps in the solution. Make sure each summary clearly describes the key points and the intermediate conclusion of the respective step in the thought process.</p> <p>Finally, your output should be in standard JSON format:</p> <pre>```json {'Thought Step 0': 'Summary of Step 0', 'Thought Step 1': 'Summary of Step 1', ...} ```</pre>

Table 13: The prompt used to summarize each step of a solution. Note that the input solution has already been decomposed into steps in a list format.

Solution Decompose Prompt

You need to read the problem-solving process below, which is divided into some steps using line breaks, with each step labeled by a tag “«»”.

[Pre Split Format Thought]

{format solution}

Then, you are required to combine consecutive steps into several logically independent sections. Note that each independent section must belong to one of the eight action categories listed below:

- (1) Problem Analysis: Decompose the original problem, clarify known conditions and objectives
- (2) Subtask Definition: Establish phased goals (e.g., equation construction, variable definition)
- (3) Calculation Solving: Perform numerical operations or symbolic derivations
- (4) Reflection Check: Verify logical/computational errors
- (5) Answer Validation: Confirm correctness of final/intermediate conclusions
- (6) Error Correction: Rectify identified issues
- (7) Strategy Shift: Switch to alternative problem-solving approach
- (8) Final Conclusion: Summarize and present ultimate answer

****Output Specifications****

- Maintain original thought chain order when outputting sections
- Section format: «start »- «end » Concise explanation the reasons for combining these steps together(max 15 words)
- Must satisfy: $start \leq end < next_start$

****Special Constraints****

- If several consecutive steps involving calculations, enumerations, proofs, or similar processes are working toward the same objective—such as solving the same equation, proposition, or problem—please group them together into a single section without splitting them apart!
- If several consecutive steps are reflections on or verifications for the same issue, combine them into a single section and do not separate them!

****Example output****

- «0 » - «2 »These three steps aim to analyze the question
- «3 » - «5 »These three steps is computing the same equation
- «6 » - «6 »This step aims to draw the final answer

Table 14: The prompt used to decompose preprocessed solutions. We define eight atomic logical actions and use them to guide the step-wise decomposition.

Match Solution and Answer Prompt

You will be given the following three sections:

[Math Problem]

{problem}

[Summarized Solution]

{summarized solution in a list format}

[Answer]

{answer of the problem in a list format}

****Task Requirements****

1. Establish a mapping relationship between all these solution steps and the answer steps.
2. For each answer step, find all solution steps whose summaries match the content of the current solution step summary.
3. Allow one-to-many or many-to-one correspondence.

****Matching Criteria****

1. The Answer Step should summarize the corresponding Solution Step, or be a refined articulation of it.
2. If the LAST step of the Solution includes the "final answer" from the Answer, it should at least match the LAST step of the Answer.

****Output Requirements****

1. Analysis Report:
 - Explain the matching basis for each solution step in order.
 - Only record clear content correspondences.
2. Mapping Table (**JSON** format):

```
{"Solution_Step_Number":  
  [Thought_Step_Numbers], ...}
```

- Unmatched thought process steps are not included.

3. Example:

```
```json
```

```
{"0": [1, 3], "1": [5]}
```

```
```
```

Table 15: The prompt used to match each answer step with its corresponding solution step(s). Note that both the solution and the answer have already been decomposed into steps in a list format.

Aware Reflection Prompt

****Instruction:****

You will receive the following two pieces of content:

[Math Problem]

{problem}

[Decomposed Solution]

{decomposed solution in a list format}

Note that this thought process has been broken down into multiple steps, documenting the complete path from the initial attempt to the final correct solution. It may include attempts with incorrect reasoning, reflections on those attempts, and subsequent corrections.

****Your Task:****

Carefully read and analyze the provided thought process to identify all segments where 'an error in previous reasoning steps was discovered through reflection or verification.' For each identified segment, complete the following tasks:

1. Record the Step Index: Indicate the step number where this segment occurs.
2. Identify the Fragment: Clearly specify the exact content of these segments.
3. Summarize the Cause: Analyze and summarize the reason why the error occurred.

****Output Format Requirements:****

Return your results in the following **JSON** format:

```json

```
{"Reflection 1": {"Step Index":
 "The step number where this segment
occurs", "Identified Fragment":
 "The exact content of the segment
where an error in previous reasoning
steps was discovered through
reflection or verification",
 "Reason": "A summary of the reason
why the error occurred"}, ...}
```
```

****Notes:****

1. The steps you find **MUST** have identified a previous **ERROR** through reflection. If this step **ONLY** involves double-checking a previous calculation to verify its correctness, without finding a specific error, then ignore this step.
2. If there are no qualifying segments in the thought process, return an empty **JSON** object: `{}`.

Table 16: The prompt used to find reflection segments within a correct solution. Note that the input solution has already been decomposed into steps in a list format.

Aware Mistake Source Prompt

You will see a math problem and a pre-decomposed thought process for solving this math problem:

[Math Problem]

{problem}

[Decomposed Solution]

{decomposed solution in a list format}

This solution records the complete mental journey from initial attempts to the final correct answer, including trials of incorrect approaches, reflections, and corrections.

The following paragraph explicitly highlights the discovery of a mistake through reflection:

[Identified Fragment]

{reflection identified fragment}

And the reason of the mistake is summarized as following:

[Reason]

{reflection reason}

Your task is to identify the earliest step in the thought process that is responsible for this mistake.

****Task Requirements****

- The step you find MUST itself actually contain logical or calculation errors. You need to verify the step you have identified CAREFULLY.
- Before stating the earliest step you believe is responsible for this mistake, you need to carefully think and provide your detailed analysis.
- Finally, based on your analysis, output the earliest thought step and step number that you believe is responsible for the mistake, and provide your reason.

****Output Format Requirements:****

Return the final results in the following JSON format:

```json

```
{"Earliest Step": "The earliest step
content that is responsible for this
mistake", "Step Index": "The step
number of this thought step",
"Reason": "The reason you believe
this step is responsible for this
mistake"}
```

```

Table 17: The prompt used to find the error source based on a reflection segment. This prompt utilizes information returned from the prompt in Table 16.

Reason All Mistakes Prompt

You will be presented with the following content:

- A math problem:
[Math Problem]
{problem}
- A decomposed thought process for solving the problem:
[Decomposed solution]
{decomposed solution in a list format}

This thought process does not directly lead to the correct answer but includes attempts with incorrect reasoning, reflections, and corrections before arriving at the right solution.

- In Step <reflection start step index> of the thought process, a critical mistake occurs:
[Earliest Step]
{reflection earliest step}
- This mistake is identified through reflection in Step <reflection step index>:
[Identified Fragment]
{reflection identified fragment}

****Task Description****

- Your task is to identify the types of thinking steps within the range from step <reflection start step index > to step<reflection step index> that meet the following definitions. You need to check each definition in sequence. Once a definition is met, immediately label it with the corresponding step type and stop checking further.
- **Reflection Steps:** A step that involves reflecting on or verifying the error and its resulting subsequent mistakes.
- **Incorrect Step:** A step is considered incorrect if: It contains mistakes itself or it continues the analysis based on the error from above steps without introducing a new, correct approach or making a proper correction.

****Specific Requirements****

- For each thinking step, carefully analyze and explain your reason before determining its type.
- The final output format should strictly adhere to standard **JSON**, for example:
```json

```
{"Step k": {"Analysis": "The specific
analysis for Step k before
determining its type",
"Step Type": ("Incorrect" /
"Reflection" / "Correct")},...}
...
```

#### **\*\*Notes\*\***

- Ensure that your analysis of each step is well-reasoned and avoids subjective assumptions.

Table 18: The prompt used to determine the category of each step within the source-to-reflection range. This prompt utilizes information returned from the prompts in Table 16 and Table 17.



### Answer Decompose Prompt

You will be presented with the following content:

**[Math Problem]**

{problem}

**[Answer]**

{answer of the problem}

#### **\*\*Task Requirements\*\***

Insert '<step>' and '</step>' into the original answer to break down the above answer into several atomic reasoning steps. Each atomic reasoning step belongs to only one of the following actions:

1. **\*\*Problem Analysis\*\***: Analyze the original problem, organize the conditions, and deepen the understanding of the problem.
2. **\*\*Set Subtask Goals\*\***: Clearly define the specific goal to be solved next, such as setting up an equation.
3. **\*\*Calculating or Solving\*\***: Calculate a variable or solve an equation.
4. **\*\*Self-Reflection\*\***: Reflect on potential mistakes and assess whether previous thoughts contain mistakes.
5. **\*\*Verify Correctness\*\***: Validate whether the final answer or intermediate conclusions are correct.
6. **\*\*Error Correction\*\***: Rectify identified mistakes.
7. **\*\*Switch Thinking\*\***: Change to a different thought for solving the problem.
8. **\*\*Arrive at the Final Answer\*\***: Conclude with the final answer.

#### **\*\*Output Format\*\***

1. The breakdown of steps should not be too coarse; each step should belong to only one type of action.
2. Separate the reasoning steps by inserting '<step>' and '</step>'.
3. Besides inserting the '<step>' and '</step>' tags, do not change the original solution.
4. Do not alter the original solution except for inserting the '<step>' and '</step>' tags; keep the original solution complete.
5. You only need to reply with the complete solution, without outputting any additional content.

Table 19: The prompt used to decompose answer. We used the same set of eight atomic logical actions as detailed in Table 14, with only minor modifications to the phrasing.