

STEPHINT: MULTI-LEVEL STEPWISE HINTS ENHANCE REINFORCEMENT LEARNING TO REASON

Kaiyi Zhang^{1,3}, Ang Lv¹, Jinpeng Li², Yongbo Wang³, Feng Wang³, Haoyuan Hu³, Rui Yan⁴

¹GSAIL, Renmin University of China, ²Peking University, ³Ant Group, ⁴SCS, Wuhan University

kzyzhang@ruc.edu.cn, ruiyan@whu.edu.cn

ABSTRACT

Reinforcement learning with verifiable rewards (RLVR) is a promising approach for improving the complex reasoning abilities of large language models (LLMs). However, current RLVR methods face two significant challenges: the near-miss reward problem, where a small mistake can invalidate an otherwise correct reasoning process, greatly hindering training efficiency; and exploration stagnation, where models tend to focus on solutions within their “comfort zone,” lacking the motivation to explore potentially more effective alternatives. To address these challenges, we propose StepHint, a novel RLVR algorithm that utilizes multi-level stepwise hints to help models explore the solution space more effectively. StepHint generates valid reasoning chains from stronger models and partitions these chains into reasoning steps using our proposed adaptive partitioning method. The initial few steps are used as hints, and simultaneously, multiple-level hints (each comprising a different number of steps) are provided to the model. This approach directs the model’s exploration toward a promising solution subspace while preserving its flexibility for independent exploration. By providing hints, StepHint mitigates the near-miss reward problem, thereby improving training efficiency. Additionally, the external reasoning pathways help the model develop better reasoning abilities, enabling it to move beyond its “comfort zone” and mitigate exploration stagnation. StepHint outperforms competitive RLVR enhancement methods across six mathematical benchmarks, while also demonstrating superior generalization and excelling over baselines on out-of-domain benchmarks.

1 INTRODUCTION

Eliciting the reasoning capabilities of large language models (LLMs) through Reinforcement Learning with Verifiable Rewards (RLVR) has emerged as a powerful paradigm (Jaech et al., 2024; Guo et al., 2025a). In RLVR frameworks, a policy model explores the solution space by generating reasoning chains. The model is then optimized using algorithms like PPO (Schulman et al., 2017) and GRPO (Shao et al., 2024), based on the advantages of final outcomes of these chains.

However, free exploration within the vast and complex solution space introduces significant challenges. A key issue is the **near-miss reward problem**, where a single incorrect step voids an otherwise reward-worthy reasoning chain. This leads to training inefficiency, as models expend resources on repeatedly almost-correct solutions. Moreover, as shown by Yue et al. (2025), existing RLVR methods often refine the model’s ability to sample known reasoning chains rather than discover novel or higher-quality ones. Consequently, when a task exceeds the model’s current capabilities, it tends to remain confined to its “comfort zone,” unable to independently advance beyond familiar solutions—an issue we term **exploration stagnation**.

We propose StepHint, a novel augmented RLVR algorithm that integrates multi-level stepwise hints to address these challenges. StepHint leverages reasoning chains from advanced models such as Deepseek-R1 (Guo et al., 2025a), partitioning them into discrete reasoning steps¹. It then provides only the initial few steps as hints for the model to complete the reasoning process. This approach effectively simplifies the solution space while preserving sufficient exploratory flexibility. Specifically, during RLVR—regardless of the policy optimization algorithm used—StepHint’s pipeline comprises two key steps:

Step 1: Adaptive Stepwise Partitioning of Reasoning Chains. Previous methods often partition reasoning chains using superficial markers such as “First” or “Step 1,” which fail to reflect the actual hierarchical structure of reasoning (Guo et al., 2025b). To overcome this, we introduce a next-token-probabilistic partitioning strategy that adaptively splits the chain into meaningful steps. Our method estimates the model’s probability of concluding

¹A *reasoning step* refers to a distinct logical stage within the overall reasoning chain and typically comprises multiple tokens. It should not be confused with a *token-prediction step* during generation or a *training step*.

the whole reasoning chain at each token—specifically, the likelihood of generating a designated end-of-reasoning token (e.g., `</think>`, which may vary by model). A token is identified as a candidate endpoint if its probability of concluding the reasoning chain exceeds the probability of concluding the reasoning chain at the next token. The intuition is that the model’s confidence in generating an end-of-reasoning token will be higher at the natural conclusion of a reasoning step than at the beginning of the subsequent step. From this set of candidates, we then randomly sample a number of points to serve as step endpoints, subject to a constraint on the minimum distance between them. These endpoints partition the entire reasoning chain into a predetermined number of steps. This enables flexible identification of coherent reasoning steps. The initial few steps are then provided as hints to guide the model’s rollouts during RL training.

Step 2: Multi-Level Hints for Problem Solving. The effectiveness of a hint depends on the number of reasoning steps it reveals. An effective hint must strike a balance—offering enough guidance, while still preserving space for free exploration. Overly detailed hints can reduce RL to a form of supervised fine-tuning (SFT), thereby limiting the model’s opportunity for independent exploration. Such exploration is crucial for activating the reasoning capabilities encoded in pre-trained models (Lv et al., 2025). Furthermore, SFT has been shown to lead to weaker generalization (Chu et al., 2025; Chen et al., 2025), particularly in training large reasoning models compared to RLVR. To formalize this trade-off, we define a hint’s “level” as the number of initial reasoning steps it provides. Under this definition, a “high-level” hint is one that contains many steps, making it highly detailed. Such a detailed, high-level hint can render the problem-solving task trivial for the model, diminishing training efficacy. Conversely, a “low-level” hint, which contains very few steps, can be insufficient to guide the model, leaving it vulnerable to the “near-miss reward problem.” Because determining the optimal hint level for a given model-problem pair is inherently difficult, StepHint generates multi-level hints for each problem. This approach ensures that, with sufficiently fine-grained step partitioning, at least one hint level is likely to be suitable for the model’s current reasoning ability.

By adaptively providing multi-level hints derived from advanced models, StepHint effectively addresses both the near-miss reward problem and exploration stagnation. First, the model receives appropriate guidance to complete reasoning chains correctly, significantly reducing near-miss rewards and improving training efficiency—leading to faster convergence. Second, exposure to high-quality hints steers the policy toward more sophisticated reasoning patterns, preventing stagnation during independent exploration. This not only enhances the model’s ability to break through its “comfort zone” but also avoids the poor generalization typical of SFT-based methods.

We evaluate StepHint by training a series of LLMs on mathematical tasks and comparing their performance against strong RLVR-enhanced baselines. Results demonstrate StepHint’s effectiveness on both in-domain (math) and out-of-domain tasks.

- **In-domain performance:** Across six math benchmarks, StepHint surpasses existing methods by an average accuracy of 3.16 percentage points. Notably, it achieves significant improvements in $\text{pass}@k$ performance—a rigorous measure of reasoning abilities (Yue et al., 2025)—on two challenging benchmarks, AIME24 and AIME25 (Li et al., 2024), even at large k values.

- **Out-of-domain generalization:** StepHint also achieves the highest results on out-of-domain, non-mathematical benchmarks such as ARC-C (Clark et al., 2018) and GPQA-D (Rein et al., 2024), highlighting its robust generalization beyond its training domain.

2 BACKGROUND: REINFORCEMENT LEARNING WITH VERIFIABLE REWARDS

The advancement of reasoning in LLMs has significantly benefited from Reinforcement Learning (RL) techniques (Hu et al., 2025; Guo et al., 2025a). These methods provide a framework for models to learn optimal reasoning chain through reward-based feedback.

Reinforcement Learning with Verifiable Rewards (RLVR) is a specialized RL paradigm highly effective for training LLMs on tasks where the correctness of an outcome can be objectively verified, such as mathematical problem-solving or code generation. In the RLVR framework, the learning process is typically driven by automated, often binary (correct/incorrect), reward signals, which facilitates scalable self-improvement (Gao et al., 2023).

2.1 POLICY OPTIMIZATION ALGORITHMS

To implement RLVR, policy optimization algorithms are used to adjust the LLM’s generation policy (π_θ) based on the reward signals. Here, we discuss the evolution from the widely-used Proximal Policy Optimization (PPO) to a more streamlined and efficient alternative, Group Relative Policy Optimization (GRPO).

Proximal Policy Optimization (PPO) PPO (Schulman et al., 2017) is a popular policy optimization algorithm. PPO aims to maximize the expected reward while preventing excessively large policy updates that could destabi-

lize the training process. Given a problem q , the policy model π_θ samples N rollouts, denoted as $\{y_1, y_2, \dots, y_N\}$. PPO optimizes the policy by maximizing the following objective function:

$$\mathcal{L}_\theta^{\text{PPO}} = \frac{1}{N} \sum_{i=1}^N \frac{1}{|y_i|} \sum_{t=1}^{|y_i|} \left\{ \min \left[r_{i,t} \hat{A}_{i,t}, \text{clip} \left(r_{i,t}, 1 - \epsilon, 1 + \epsilon \right) \hat{A}_{i,t} \right] \right\} - \beta D_{KL}(\pi_\theta || \pi_{\text{ref}}),$$

where:

- $r_{i,t} = \frac{\pi_\theta(y_{i,t}|q, y_{i,<t})}{\pi_{\text{old}}(y_{i,t}|q, y_{i,<t})}$ is the probability ratio between the current policy π_θ and the policy before the update, π_{old} . The clip function bounds this ratio, which disincentivizes overly aggressive policy changes that could destabilize training.
- $\hat{A}_{i,t}$ is the advantage of taking token $y_{i,t}$ as the t token of rollout i . It quantifies how much better that action is compared to the average action value at that state.
- $\beta D_{KL}(\pi_\theta || \pi_{\text{ref}})$ is a penalty term that discourages the current policy from deviating too far from a reference policy π_{ref} (often the initial model).

In standard PPO implementation, calculating the advantage $\hat{A}_{i,t}$ for each token requires a critic model. This critic learns to estimate the value of states, often in conjunction with Generalized Advantage Estimation (GAE) (Schulman et al., 2015b), to compute the per-token advantages. However, PPO requires training a reliable critic model, which can be computationally expensive and complex. This can lead to “reward hacking,” where the policy model learns to exploit the critic’s inaccuracies to maximize rewards without achieving the actual task goal (Amodio et al., 2016).

Group Relative Policy Optimization (GRPO) GRPO (Shao et al., 2024), a simpler yet effective alternative, was developed to address the challenges of PPO. GRPO has gained significant traction for its remarkable performance and efficiency in complex reasoning tasks, especially mathematical reasoning tasks (Liu et al., 2025; Yan et al., 2025; Zeng et al., 2025; Hu et al., 2025).

The core innovation of GRPO is to bypass the need for a critic model and per-token advantage calculation entirely. Instead, it computes a single, uniform advantage value for all tokens within a rollout, based on the final outcome of that entire rollout.

Specifically, for a given problem q , N rollouts $\{y_1, y_2, \dots, y_N\}$ are sampled. Each complete rollout y_i is assigned a final reward $R(y_i)$, which is typically binary in RLVR settings: $R(y_i) = 1$ if the answer of y_i is correct, and $R(y_i) = 0$ otherwise. GRPO then calculates a rollout-level advantage by normalizing this reward across the group. This advantage value is assigned to every token within that rollout:

$$\hat{A}_{i,t}^{\text{GRPO}} = \frac{R(y_i) - \text{mean}(\{R(y_1), \dots, R(y_N)\})}{\text{std}(\{R(y_1), \dots, R(y_N)\})}. \quad (1)$$

This rollout-level advantage $\hat{A}_{i,t}^{\text{GRPO}}$ then replaces the complex, per-token advantage $\hat{A}_{i,t}$ within the PPO objective function. The core clipping mechanism and KL-divergence penalty of PPO are retained, but the need for a separate critic model is eliminated entirely, drastically simplifying the training pipeline and reducing computational overhead.

The field of policy optimization includes other notable algorithms such as Trust Region Policy Optimization (TRPO) (Schulman et al., 2015a), REINFORCE (Sutton et al., 1999) and Asynchronous Advantage Actor-Critic (A3C) (Mnih et al., 2016), PPO and GRPO have emerged as the most widely adopted methods for training LLMs on reasoning tasks. Therefore, the discussions and experiments in this paper will focus on these two algorithms as they represent the predominant approaches in this area.

3 METHOD

We conceptualize the reasoning process as a stepwise simplifying of a solution space. This perspective inspires our core idea: by providing stepwise hints, we can guide the model’s exploration toward more promising directions. To establish a foundation, Section 3.1 first formalizes the generation of reasoning chains in LLMs from a view of solution space exploration. This formalization not only helps identify critical issues in existing methods, such as the near-miss reward problem and exploration stagnation, but also aids in better understanding our method, StepHint (Section 3.2), which is specifically designed to address these challenges.

3.1 PRELIMINARIES: A SOLUTION SPACE VIEW OF REASONING

Autoregressive LLMs generate reasoning chains by producing tokens sequentially, where each new token depends on the preceding ones (Radford et al., 2019; Grattafiori et al., 2024). This iterative simplification process is an

exploration within a solution space \mathcal{R} . The space \mathcal{R} consists of all possible reasoning chains the model might generate given a prompt \mathcal{C} (Guo et al., 2025a). As tokens are generated, each token decision prunes the solution space, reducing the ambiguity and complexity of the final output.

To formalize this exploration process, we model it as a sequence of states. Each state, S_k , represents the partial reasoning chain constructed from the first k generated tokens. The transition from one state to another corresponds to generating a new token. We quantify the complexity of the solution space at each state using conditional entropy $H(\mathcal{R}|S_k)$, which measures the remaining uncertainty given the tokens generated so far.

1. **Initial State** (S_0): The process begins with the prompt \mathcal{C} , which typically defines the problem to be solved in RLVR. This prompt constitutes the initial state, $S_0 = \mathcal{C}$. The initial complexity of the solution space, given only the prompt, is quantified by the conditional entropy $H(\mathcal{R}|\mathcal{C})$ (Shannon, 1948). As the solution space is composed of discrete token sequences, this is computed as:

$$H(\mathcal{R}|\mathcal{C}) = - \sum_{r \in \mathcal{R}} p(r|\mathcal{C}) \log p(r|\mathcal{C}).$$

It is important to note that this formulation serves as a theoretical model for qualitative analysis. A direct quantitative computation of this entropy is generally intractable, as it would require summing over the entire space of all possible reasoning chains \mathcal{R} . Despite its intractability, this entropy-based framework provides a powerful conceptual tool for analysis.

A high value of $H(\mathcal{R}|\mathcal{C})$ indicates a complex and unconstrained solution space, while a low value suggests that the prompt has already constrained the problem to a narrower range.

2. **Intermediate States** (S_k): After generating k tokens, t_1, \dots, t_k , the system transitions to an intermediate state $S_k = (\mathcal{C}, t_1, \dots, t_k)$. The remaining complexity of the solution space, given the generated tokens, is captured by the conditional entropy $H(\mathcal{R}|S_k) = H(\mathcal{R}|\mathcal{C}, t_1, \dots, t_k)$.

The following Proposition formalizes that, in expectation, each token-prediction step reduces or maintains the entropy, which corresponds to the complexity of the solution space, often stated as “conditioning reduces entropy.” (Cover, 1999)

Proposition 1. *Let \mathcal{R} be the solution space and S_{k-1} be the state after $k-1$ tokens have been generated. Upon generating the next token t_k to form state $S_k = (S_{k-1}, t_k)$, the expected entropy of the solution space is bounded by the current entropy:*

$$\mathbb{E}_{t_k \sim P(\cdot|S_{k-1})} [H(\mathcal{R}|S_k)] \leq H(\mathcal{R}|S_{k-1}).$$

We leave the detailed proof in Appendix A.1. Proposition 1 provides a formal foundation for understanding autoregressive generation as a structured exploration. This process incrementally refines a vast solution space, converging on a specific output by reducing uncertainty and complexity at each step. It should be noted that, the reduction in entropy quantifies the convergence of certainty, not necessarily the correctness or logical validity of the reasoning chain. A model can become increasingly certain about a flawed or nonsensical conclusion, and this would still manifest as a decrease in entropy. Therefore, entropy reduction should be understood as a necessary, but not sufficient, condition for reasoning. This view reveals a failure mode in reasoning, occurring when the model makes a critical early error. This misstep irrevocably prunes the subspace of correct solutions, R^* . Formally, this corresponds to a state S_k where the probability of the correct solution set collapses to zero: $P(R^*|S_k)$. The model may continue reducing entropy confidently, but within an incorrect subspace $R \setminus R^*$, leading to a “confident but wrong” conclusion.

3.2 STEPHINT: MULTI-LEVEL STEPWISE HINTS ENHANCE RLVR

Based on the views above, the reasoning efforts following the initial error might be logical and proper, but the answer remains wrong because they are based on an incorrect premise. This near-miss reward issue could be avoided if slight guidance or supervision were provided at the beginning. Meanwhile, the exploration of the solution space is largely constrained by the model’s ability. Without external guidance, the exploration will be limited to a narrow subspace (Yue et al., 2025). In this section, we introduce how our proposed RLVR enhancement method, StepHint, addresses these two issues by providing strong reasoning chains as hints to models during training.

Given a problem, StepHint first obtains a valid reasoning chain from a stronger model, and then performs two key stages to enhance the target model being trained: (1) adaptive stepwise partitioning of on-hand reasoning chains and (2) multi-level hints for problem solving. The on-hand reasoning chains are collected offline by querying a stronger model before training the target model and retaining only outputs that have been verified as correct.

In the following, we focus on detailing the latter two stages. We will frequently use the term “reasoning step” to refer to an intact unit of thought, which may consist of several sentences. Please note that this is distinct from a “training step,” which refers to updating the model after processing a batch of data, or a “next-token prediction step,” which generates a single token at a time.

3.2.1 ADAPTIVE STEPWISE PARTITIONING OF ON-HAND REASONING CHAINS

Definitions Let the reasoning chain be denoted as $G = t_1 \circ t_2 \circ \dots \circ t_n$, where each t_i represents a single token, and \circ denotes concatenation. A *reasoning step* corresponds to a sequence of tokens $t_i \circ \dots \circ t_j$ ($1 \leq i \leq j \leq n$) that forms an intact unit of thought. A *hint* is composed of one or more reasoning steps and serves as a conditioning prompt that guides the target model’s reasoning toward a promising direction, helping it explore otherwise intractable solution spaces. The *level of a hint* is determined by the number of reasoning steps it contains—the more reasoning steps included, the richer the guidance it offers to the target model, and thus the higher its level.

Method Details We need a flexible method to adaptively partition a complete reasoning chain G into m reasoning steps, then combine appropriate number of reasoning steps as a hint in appropriate level to the target model. Conventional approach relies on syntactic cues, such as keywords like “first,” “next,” or “Step 1.” However, such heuristics are prone to misidentifying the boundaries of reasoning steps and lack the flexibility. (Moen, 2017)

In contrast, we leverage the model’s output probability distribution to identify the boundaries of reasoning steps. We hypothesize that when a reasoning step concludes, the model’s perceived probability of completing the entire chain should be relatively high. Conversely, at the beginning of a new step, this probability should decrease as the model expects additional reasoning to follow. This perceived likelihood of reasoning completion can be captured by the probability assigned to a special “end-of-thinking” token, $\langle \text{think} \rangle$, which is explicitly introduced during pretraining to mark the conclusion of a reasoning step (Team, 2024; Guo et al., 2025a). Formally, the model’s tendency to conclude a reasoning step at token t_i can be quantified by the probability $p(\langle \text{think} \rangle | G_i)$, where G_i denotes the token sequence up to t_i .

This hypothesis leads us to our core partitioning method: a token t_i is considered as a candidate reasoning step boundary if and only if the probability of concluding the reasoning chain after t_i is greater than the probability of concluding it after the subsequent token, t_{i+1} : $p(\langle \text{think} \rangle | G_i) > p(\langle \text{think} \rangle | G_{i+1})$. By iterating through the entire reasoning chain, we collect all tokens satisfying this condition as candidate boundaries. To maintain high-quality reasoning steps, we enforce two constraints during partitioning: (1) adjacent boundaries must be at least l tokens apart to avoid overly short steps, and (2) the number of steps must be equal to the predetermined value, m . In practice, multiple valid partitionings may satisfy these constraints, we randomly sample one to proceed with. Figure 1 illustrates this token-distribution-based partitioning method.

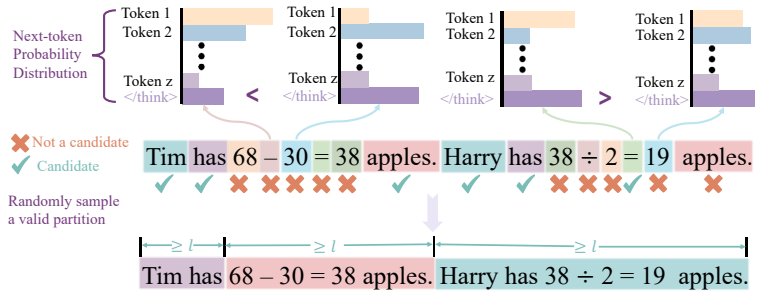


Figure 1: Adaptive stepwise partitioning of a reasoning chain: step boundaries are identified where the probability of concluding the reasoning chain after the current token, $p(\langle \text{think} \rangle | G_i)$, is greater than concluding after the subsequent token, $p(\langle \text{think} \rangle | G_{i+1})$. A final partition is chosen to meet constraints on step count m and minimum length l .

3.2.2 MULTI-LEVEL HINTS FOR PROBLEM SOLVING

Building on the adaptive stepwise partitioning method described above, we divide the reasoning chain into m reasoning steps. A key question is how many of these steps should be included as a complete hint for the target model.

An ideal level of hinting matches the model’s current capabilities at each stage of training—it is neither too weak nor so strong that it simplifies the task. Moreover, this optimal level shifts continuously as the model’s reasoning ability evolves. Considering these challenges, rather than determining the ideal hint level at each training step, we provide hints at multiple levels. Our core hypothesis is that if the partitioning is fine-grained enough, there is very likely to be a hint that fits the model’s needs well.

Specifically, we construct a set of $m - 1$ multi-level hints, \mathcal{H} . Each hint is a prefix of the full reasoning chain G , created by concatenating the first j steps:

$$\mathcal{H} = \{h_j | h_j = S_1 \circ S_2 \circ \dots \circ S_j, \text{ for } j = 1, \dots, m - 1\},$$

where S_i represents the i -th reasoning step. Low-level hints preserve considerable problem-solving difficulty. In contrast, high-level hints significantly simplify the solution space, making the completion easier.

For each problem in the training set, we construct three types of prompts for the model to complete:

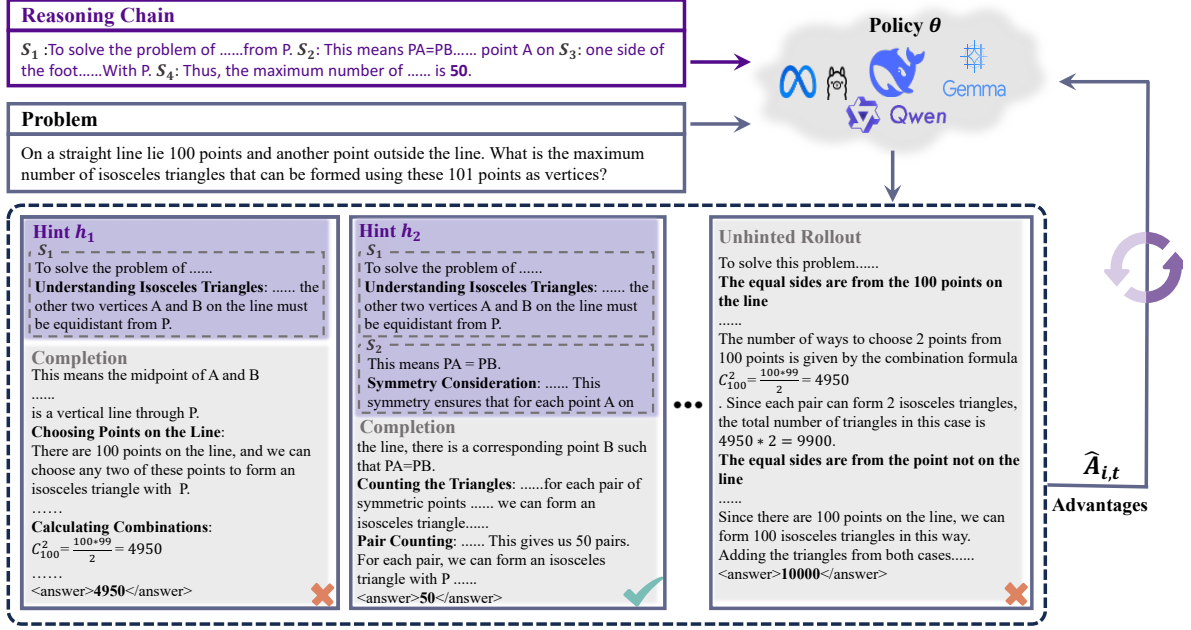


Figure 2: An overview of the multi-level hinting process. The process begins with a ground-truth reasoning chain, which is partitioned into m steps (Section 3.2.1). From these steps, we construct $m - 1$ prefix-based hints (h_1, h_2, \dots, h_{m-1}). The model is trained to generate completions from each hint level, as well as from scratch (Unhinted), and a reference trajectory.

1. **Hinted Problems:** For each of the $m - 1$ hint levels, the model is asked to complete the reasoning from that hint using k_{hint} rollout attempts per hint.
2. **Unhinted Problems:** To preserve the model’s independent exploration, it also solves the problem from scratch without any hints. It is allowed k_{unhint} rollouts in this case.
3. **Reference Trajectory:** The original ground-truth reasoning chain G is also provided to the target model and used to assign rewards. This ensures that the model is consistently exposed to the correct solution path during training.

Figure 2 illustrates this multi-level hinting and completion process. In total, the model generates $k_{\text{hint}}(m - 1) + k_{\text{unhint}} + 1$ completions per training problem, each receiving a reward based on correctness. StepHint strikes a balance between guiding the model with reliable hints and allowing it to learn from its own exploration mistakes, leading to more effective learning.

The above method applies to most RLVR algorithms but poses issues for GRPO (He et al., 2025), prompting further adaptations and discussion. In GRPO, an incorrect completion assigns negative advantages to all tokens in the rollout—including those in the correct hint prefix. This steers the model away from the correct reasoning chains. To address this, we modify GRPO by clipping negative advantages to zero for tokens in the hint prefix when the completion is incorrect; that is, we set $\hat{A}_{i,t}^{\text{GRPO}} = \max(0, \hat{A}_{i,t}^{\text{GRPO}})$ (see Eq. 1 for the definition of $\hat{A}_{i,t}^{\text{GRPO}}$). This adaptation prevents the model from being penalized for correct prefixes, aligning the training process with our intended mechanism. We demonstrate its empirical effectiveness in the next section.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETTINGS

Training Data To construct a scenario where the training problems are particularly challenging for the target model to independently explore, we gather difficult problems from the DAPO dataset (Yu et al., 2025) and selected problems of difficulty level 7 or higher from the DEEPMATH dataset (He et al., 2025). The DEEPMATH dataset provides solution reasoning chains, while we generate reasoning chains for the DAPO dataset using the DAPO-Qwen-32B (Yu et al., 2025), QWQ-32B (Team, 2025), and DeepSeek-R1-Distill-Qwen-32B (Guo et al., 2025a) models, as described in Appendix A.3. To ensure data quality, we filter out excessively long answers, retaining

only those with a length of 4,096 tokens or fewer. All reasoning chains are partitioned into $m = 4$ steps, each longer than $l = \frac{L}{8}$ tokens, where L is the total length of a reasoning chain G , with QWQ-32B (Team, 2025). This threshold ensures that even the shortest step contains a substantive amount of information, while allowing for the natural length variation between different steps in a complex reasoning process.

This results in a training dataset of approximately 26,000 instances.

Hyperparameters We train two backbone models for 5 epochs each: Qwen-2.5-7B-Instruct (Yang et al., 2024a; Team, 2024) and Qwen-2.5-Math-7B (Yang et al., 2024b). The training prompt template is shown in Appendix A.2. The training is based on the VeRL framework (Sheng et al., 2024). We set a global batch size of 128 and a fixed learning rate of $1e - 6$. Following (Yan et al., 2025), we set the KL loss coefficient $\beta = 0$, indicating no reference model is used for regularization. We set $k_{\text{hint}} = 2$, and $k_{\text{unhint}} = 5$. During training, the temperature for rollout generation is set to 1.0.

Baselines We compare against three categories of baselines:

1. **Vanilla GRPO:** The model is trained using vanilla GRPO on Qwen-2.5-7B-Instruct with the same settings as StepHint.
2. **SFT:** We also fine-tune the models with available reasoning chains using supervised fine-tuning (SFT), applying the same learning rate and training epochs as StepHint.
3. **Other RLVR Enhancement Methods:** We evaluate other RLVR enhancement techniques, including: SimpleRL-Zero-7B (Zeng et al., 2025), Qwen-2.5-Math-7B-SimpleRL-Zoo, OpenReasonerZero-7B (Hu et al., 2025), Oat-7B (Liu et al., 2025), Luffy-Qwen-2.5-7B-Instruct (Yan et al., 2025), and Luffy-Qwen-Math-7B-Zero. Evaluations are conducted using their publicly released model weights.

Evaluation We follow prior work (Yan et al., 2025; Liu et al., 2025) and evaluate on six math datasets: AIME 2024, AIME 2025, AMC (Li et al., 2024), Minerva (Lewkowycz et al., 2022), OlympiadBench (He et al., 2024), and MATH500 (Hendrycks et al., 2021). For the AIME 24/25 and AMC datasets, given their limited data points, we conduct each evaluation five times and report the average results. To evaluate generalization, we also incorporate two non-math benchmarks, ARC-C (Clark et al., 2018) and GPQA-Diamond (Rein et al., 2024), as out-of-domain tests for models trained on math problems. We report the weighted average accuracy for both in-domain and out-of-domain benchmarks. The generation length is also set to 4,110. All results were evaluated using OAT-Grade (Liu et al., 2024).

4.2 MAIN RESULTS

Table 1 shows the overall performance of StepHint and baseline methods.

When applied to the general-purpose model, Qwen-2.5-7B-Instruct, StepHint achieves the highest performance on in-domain math tasks among all compared methods. Compared to other RLVR methods, StepHint shows a 7 percentage point improvement over the next-best method, LUFFY. Furthermore, StepHint consistently surpasses the SFT baseline, indicating that StepHint effectively learns beyond simple token imitation, leading to improved reasoning outcomes. Notably, the Qwen-2.5-7B-Instruct model trained with StepHint outperforms the specialized Qwen-2.5-Math-7B fine-tuned with any other RLVR method, highlighting the substantial boost in reasoning ability provided by StepHint and allowing a generalist model to outperform a specialist in its own domain.

For the specialized Qwen-2.5-Math-7B model, as expected from its specialized design (Yang et al., 2024b), the Math model performs lower on the out-of-domain non-math benchmarks compared to the general-purpose Qwen-2.5-7B-Instruct. However, StepHint not only leads the board in in-domain math tasks compared with baselines but also enables the Math model to achieve the highest out-of-domain test performance among all baselines. This suggests that the improvements are not solely due to domain-specific knowledge but may also reflect an enhancement of the model’s general reasoning capabilities.

4.3 PASS@K EVALUATION

Many studies (Chen et al., 2021; Wang et al., 2022) show that with a limited number of rollouts, models may perform poorly on certain tasks. However, with a sufficiently large number of rollouts, they are more likely to sample good solutions for solving these problems. Therefore, to fully assess the model’s potential performance, passk accuracy (where k is very large) is a more suitable metric (Yue et al., 2025). In this context, a problem is considered solved if any of the k sampled reasoning chains yields a correct answer.

Figure 3 presents the pass@k results on the AIME24 and AIME25 datasets. The results demonstrate that StepHint improves the model’s pass@k performance as k increases. In contrast, Vanilla-GRPO shows a slower rate of improvement at higher values of k , which aligns with findings from previous work (Yue et al., 2025). The superior

Table 1: Performance comparison of StepHint with baseline methods on in-domain and out-of-domain benchmarks. The top score in each column is in **bold**, and the second-highest is underlined. Backbone models are denoted by: *Qwen-2.5-7B-Instruct, †Qwen-2.5-7B, ‡Qwen-2.5-Math-7B.

Model	In-Domain					Out-of-Domain				
	AIME24 (avg@5)	MATH500 (pass@1)	AMC (avg@5)	Olympiad (pass@1)	Minerva (pass@1)	AIME25 (avg@5)	Avg. -	ARC-C (pass@1)	GPQA-D (pass@1)	Avg. -
SFT*	20.00	78.80	53.73	36.89	36.40	10.67	53.76	90.96	23.23	83.28
SFT‡	26.00	82.20	59.52	45.19	34.19	15.33	54.77	67.66	23.74	61.31
On-policy RLVR Replication										
Vanilla-GRPO*	24.67	76.60	51.33	43.41	<u>39.34</u>	10.67	52.59	<u>91.30</u>	37.37	<u>83.51</u>
Other RLVR Methods										
ORZ-7B†	24.67	81.00	46.90	45.60	33.46	15.30	53.76	90.53	<u>40.40</u>	83.28
SimpleRL†	22.00	76.00	47.90	39.30	36.40	5.30	49.83	74.74	32.32	68.61
SimpleRL‡	28.00	76.20	57.59	37.93	34.93	12.00	49.80	63.91	27.27	58.61
Oat‡	36.00	78.40	59.75	42.52	36.40	10.00	52.92	59.89	33.84	56.13
LUFFY*	21.30	77.80	44.82	40.00	36.40	14.67	50.69	81.83	32.32	74.67
LUFFY‡	27.33	<u>83.20</u>	60.24	<u>48.00</u>	38.97	<u>17.33</u>	57.19	81.83	35.86	75.19
StepHint*	<u>29.33</u>	82.80	<u>61.69</u>	47.41	43.38	17.30	<u>57.69</u>	91.89	42.42	84.74
StepHint‡	36.00	87.00	62.65	52.15	38.24	18.87	60.35	84.73	38.89	78.10

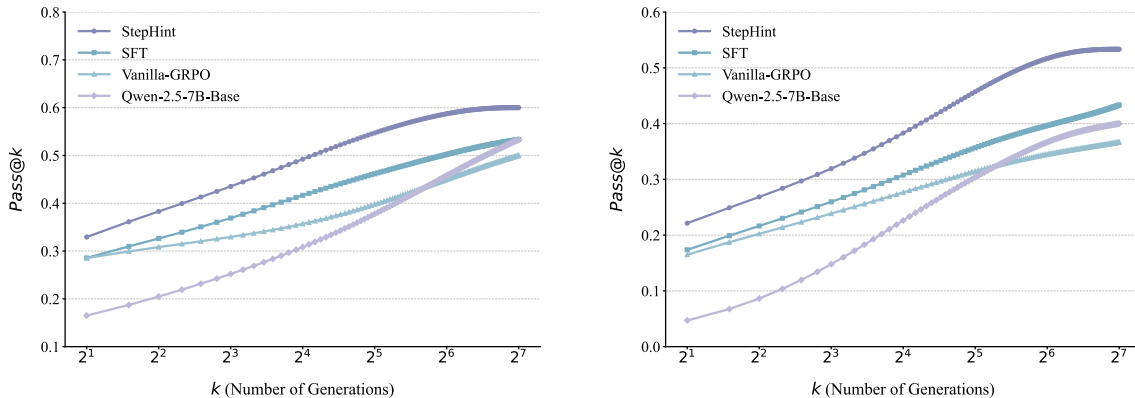


Figure 3: Comparison of pass@k results on the AIME24 and AIME25 datasets. **Left:** AIME24; **Right:** AIME25.

performance under pass@k evaluation further validates the effectiveness of StepHint. Additionally, the performance difference can be attributed to the exploration strategies the models employ. While Vanilla-GRPO suffers from “exploration stagnation,” StepHint guides the model’s exploration, helping it break free from its “comfort zone.”

4.4 METHOD ANALYSIS FROM TRAINING DYNAMICS

We analyze the training dynamics by comparing StepHint and Vanilla-GRPO on three key metrics: entropy, response length, and training rewards, as shown in Figure4. The differences in these metrics offer valuable insights into the underlying behavior and effectiveness of StepHint.

Reward The reward curves highlight the different learning phases. Due to the multi-level stepwise hints provided by StepHint, the problem-solving difficulty for the model is lower compared to Vanilla-GRPO. As a result, the reward score for StepHint is consistently higher, reflecting the mitigation of the near-miss reward issue.

A closer examination of the trends in both curves offers further insights. Vanilla-GRPO shows a steady, monotonic increase in reward as it refines its existing policy. In contrast, StepHint experiences a brief initial dip in reward before a rapid and sustained increase. This initial dip likely reflects an adaptation period where the model transi-

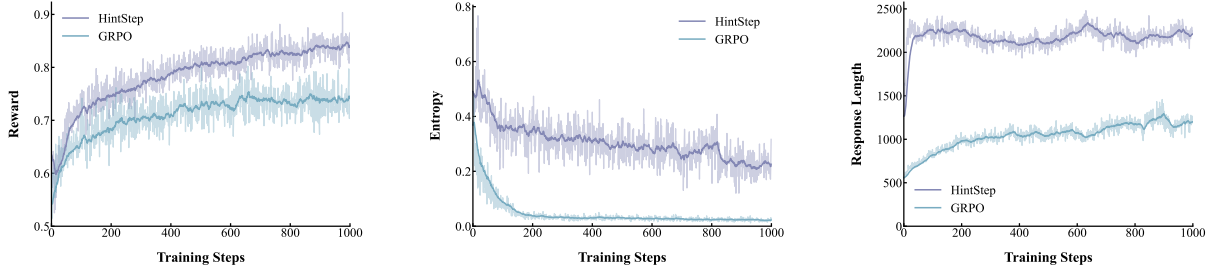


Figure 4: Training dynamics of StepHint compared with GRPO. **Left:** Reward; **Middle:** Entropy; **Right:** Response Length.

tions from simple exploitation to a more complex, hint-guided exploration. Once adapted, the model efficiently discovers higher-reward solutions, leading to faster and effective learning to reason.

Entropy Both methods exhibit an overall decrease in entropy, though their trajectories diverge as training progresses. The policy entropy for StepHint remains consistently higher than that of Vanilla-GRPO. This suggests that the hints provided by StepHint encourage a more diverse policy, preventing premature convergence to a narrow solution subspace and promoting a higher level of exploration (Cheng et al., 2025). This trend reflects, to some extent, the mitigation of exploration stagnation.

Response Length The two methods demonstrate distinct patterns in generated response length. StepHint shows an initial sharp increase in length, which we attribute to the model learning to mimic the structured, stepwise reasoning chains provided by the multi-level hints. These hints are often more detailed than the model’s initial, more direct responses.

These dynamics collectively illustrate that StepHint fosters a more effective process for developing reasoning abilities.

5 RELATED WORKS

RL-based post-training has demonstrated remarkable success in mathematical reasoning tasks (Shao et al., 2024; Yang et al., 2024b). Research in this area has primarily advanced in three directions: (1) optimizing the models and their training data, (2) refining inference-time strategies, and (3) improving policy optimization methods.

The first direction involves constructing high-quality, large-scale mathematical reasoning datasets (Wang et al., 2023; Ye et al., 2025) and designing specialized training or fine-tuning methods (Jaech et al., 2024; Mitra et al., 2024). The second direction focuses on guiding the model’s step-by-step thought processes without altering its underlying weights, typically through sophisticated prompting techniques such as Chain-of-Thought (Wei et al., 2022) and innovations in in-context learning (Wu et al., 2024). The third direction aims at developing advanced policy optimization algorithms. GRPO, an advancement of PPO (Schulman et al., 2017), has recently gained popularity due to its simplicity and strong performance (Hu et al., 2025; Zeng et al., 2025). Additionally, several improvements have been proposed for GRPO; for example, Liu et al. (2025) identified inherent length and difficulty biases in vanilla GRPO and addressed these issues.

(Yan et al., 2025) is also related to this work. The authors use an *entire* reasoning chain from stronger models as a reference trajectory. The reference is provided to the target model, which then independently generates rollouts several times without any hints. As a result, this approach does not fully address the near-miss reward issue, as the model still independently explores most of the time. While using reasoning chains from stronger models may help mitigate exploration stagnation, it inherently undermines the core exploration mechanism of the RL algorithm. In contrast, StepHint better combines model-independent exploration with external hints, leading to more effective learning.

6 CONCLUSION

In this paper, we introduce StepHint, a novel RLVR algorithm that incorporates multi-level stepwise hints. This mechanism is designed to provide the model with assistance tailored to its evolving capabilities, thereby facilitating the learning process by addressing challenges such as near-miss rewards and exploration stagnation. StepHint not only outperforms strong baselines on mathematical benchmarks but also demonstrates robust generalization to out-of-domain tasks, highlighting the promising potential of the stepwise hinting paradigm for RLVR enhancement.

REFERENCES

- Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.
- Hardy Chen, Haoqin Tu, Fali Wang, Hui Liu, Xianfeng Tang, Xinya Du, Yuyin Zhou, and Cihang Xie. Sft or rl? an early investigation into training rl-like reasoning large vision-language models. *arXiv preprint arXiv:2504.11468*, 2025.
- Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.
- Daixuan Cheng, Shaohan Huang, Xuekai Zhu, Bo Dai, Wayne Xin Zhao, Zhenliang Zhang, and Furu Wei. Reasoning with exploration: An entropy perspective. *arXiv preprint arXiv:2506.14758*, 2025.
- Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, Dale Schuurmans, Quoc V Le, Sergey Levine, and Yi Ma. Sft memorizes, rl generalizes: A comparative study of foundation model post-training. *arXiv preprint arXiv:2501.17161*, 2025.
- Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.
- Thomas M Cover. *Elements of information theory*. John Wiley & Sons, 1999.
- Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In *International Conference on Machine Learning*, pp. 10835–10866. PMLR, 2023.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-rl: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025a.
- Yiran Guo, Lijie Xu, Jie Liu, Dan Ye, and Shuang Qiu. Segment policy optimization: Effective segment-level credit assignment in rl for large language models. *arXiv preprint arXiv:2505.23564*, 2025b.
- 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.
- Zhiwei He, Tian Liang, Jiahao Xu, Qiuzhi Liu, Xingyu Chen, Yue Wang, Linfeng Song, Dian Yu, Zhenwen Liang, Wenxuan Wang, et al. Deepmath-103k: A large-scale, challenging, decontaminated, and verifiable mathematical dataset for advancing reasoning. *arXiv preprint arXiv:2504.11456*, 2025.
- Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.
- Jingcheng Hu, Yinmin Zhang, Qi Han, Daxin Jiang, Xiangyu Zhang, and Heung-Yeung Shum. Open-reasoner-zero: An open source approach to scaling up reinforcement learning on the base model. *arXiv preprint arXiv:2503.24290*, 2025.
- Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.
- Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35:3843–3857, 2022.
- 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, 2024.
- Zichen Liu, Changyu Chen, Xinyi Wan, Chao Du, Wee Sun Lee, and Min Lin. Oat: A research-friendly framework for llm online alignment. <https://github.com/sail-sg/oat>, 2024.

- Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding rl-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025.
- Ang Lv, Ruobing Xie, Xingwu Sun, Zhanhui Kang, and Rui Yan. The climb carves wisdom deeper than the summit: On the noisy rewards in learning to reason, 2025. URL <https://arxiv.org/abs/2505.22653>.
- David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.
- Arindam Mitra, Hamed Khanpour, Corby Rosset, and Ahmed Awadallah. Orca-math: Unlocking the potential of slms in grade school math. *arXiv preprint arXiv:2402.14830*, 2024.
- Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937. PmLR, 2016.
- Marie-Francine Moens. Argumentation mining: How can a machine acquire common sense and world knowledge? *Argument & Computation*, 9:1 – 14, 2017. URL <https://api.semanticscholar.org/CorpusID:3483942>.
- Yury Polyanskiy and Yihong Wu. *Information theory: From coding to learning*. Cambridge university press, 2025.
- Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.
- 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.
- John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pp. 1889–1897. PMLR, 2015a.
- John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015b.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- Claude E Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.
- Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.
- Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.
- Qwen Team. Qwen2.5: A party of foundation models, September 2024. URL <https://qwenlm.github.io/blog/qwen2.5/>.
- Qwen Team. Qwq-32b: Embracing the power of reinforcement learning, March 2025. URL <https://qwenlm.github.io/blog/qwq-32b/>.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.
- Zengzhi Wang, Rui Xia, and Pengfei Liu. Generative ai for math: Part i–mathpile: A billion-token-scale pretraining corpus for math. *arXiv preprint arXiv:2312.17120*, 2023.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Jinyang Wu, Mingkuan Feng, Shuai Zhang, Feihu Che, Zengqi Wen, Chonghua Liao, and Jianhua Tao. Beyond examples: High-level automated reasoning paradigm in in-context learning via mcts. *arXiv preprint arXiv:2411.18478*, 2024.

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

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zhihao Fan. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024a.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024b.

Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. Limo: Less is more for reasoning. *arXiv preprint arXiv:2502.03387*, 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.

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

A APPENDIX

A.1 PROOF OF PROPOSITION 1

Proposition 1:

$$\mathbb{E}_{t_k \sim p(\cdot | S_{k-1})} [H(\mathcal{R} | S_k)] \leq H(\mathcal{R} | S_{k-1})$$

Proof. We want to prove the following inequality:

$$\mathbb{E}_{t_k \sim p(\cdot | S_{k-1})} [H(\mathcal{R} | S_k)] \leq H(\mathcal{R} | S_{k-1})$$

This inequality states that the expected entropy of solution space \mathcal{R} conditioned on the state S_k is less than or equal to the entropy of \mathcal{R} conditioned on the prior state S_{k-1} . The state S_k is reached from S_{k-1} after an observation or transition t_k . Let's denote the random variable for this transition as T_k .

First, let's clarify the notation. The expression on the left-hand side, $\mathbb{E}_{t_k \sim p(\cdot | S_{k-1})} [H(\mathcal{R} | S_k)]$, represents the conditional entropy $H(\mathcal{R} | S_k)$. The conditional entropy $H(X | Y)$ is defined as the expectation of the entropy of X over the values of Y . The subscript simply makes the underlying probability model explicit: the distribution of the state S_k is induced by the distribution of the prior state S_{k-1} and the transition T_k .

The proof of this relationship relies on the non-negativity of conditional mutual information. The conditional mutual information between two random variables, \mathcal{R} and T_k , given a third variable S_{k-1} , is defined as:

$$I(\mathcal{R}; T_k | S_{k-1}) = H(\mathcal{R} | S_{k-1}) - H(\mathcal{R} | S_{k-1}, T_k)$$

A fundamental property of mutual information is that it is always non-negative (MacKay, 2003; Polyanskiy & Wu, 2025):

$$I(\mathcal{R}; T_k | S_{k-1}) \geq 0$$

From this, it directly follows that:

$$H(\mathcal{R} | S_{k-1}) \geq H(\mathcal{R} | S_{k-1}, T_k)$$

This equation shows that conditioning on an additional variable, T_k , can only decrease (or leave unchanged) the entropy of \mathcal{R} .

Now, we must relate the term $H(\mathcal{R}|S_{k-1}, T_k)$ to the term $H(\mathcal{R}|S_k)$. The state S_k is the result of a process that starts in state S_{k-1} and undergoes the transition T_k . This means that the information contained in the pair of variables (S_{k-1}, T_k) fully determines the state S_k . In many typical models, knowing S_k is equivalent to knowing the pair (S_{k-1}, T_k) that produced it. If we assume this equivalence, then conditioning on S_k is the same as conditioning on the pair (S_{k-1}, T_k) . Therefore, we have:

$$H(\mathcal{R}|S_k) = H(\mathcal{R}|S_{k-1}, T_k)$$

Substituting this equality back into our previous inequality, we arrive at the desired result:

$$H(\mathcal{R}|S_k) \leq H(\mathcal{R}|S_{k-1})$$

This completes the proof. □

A.2 TEMPLATE

Template

```
<|im.start|>system
You are a helpful assistant. The assistant first thinks about the reasoning process in the mind
and then provides the user with the answer. The reasoning process and answer are enclosed
within <think> </think> and <answer> </answer> tags, respectively, i.e., <think>
reasoning process here </think><answer> answer here </answer>. Now the user asks you
to solve a mathematical reasoning problem. After thinking, when you finally reach a solution,
clearly state the answer marked with \boxed{} and within <answer> </answer> tags, i.e.,
<answer>\boxed{answer}</answer>
<|im.end|>
<|im.start|> user
{question}
<|im.end|>
<|im.start|>assistant
<think>
```

A.3 TRAINING DATA CONSTRUCTION

For each question in the DAPO dataset (Yu et al., 2025), we sample a total of 12 reasoning chains using DAPO-Qwen-32B (Yu et al., 2025), QWQ-32B (Team, 2025), and DeepSeek-R1-Distill-Qwen-32B (Guo et al., 2025a), with 4 samples per model. The sampling was conducted under a 0-shot setting, with a temperature of 1 and a maximum length of 8,192. We filter these to retain all reasoning chains that are both correct and have a length of no more than 4,110. In cases where multiple chains satisfy these conditions, we randomly select one.