# Towards Hierarchical Multi-Step Reward Models for Enhanced Reasoning in Large Language Models

**Teng Wang[1], Zhangyi Jiang[1], Zhenqi He[1], Hailei Gong[2*], Shenyang Tong[1], Wenhan Yang[1], Zeyu Li[3], Yanan Zheng[4], Zifan He[1], Zewen Ye[5], Shengjie Ma[6*] Jianping Zhang[7*]**

[1] the University of Hong Kong, [2] Tsinghua University,
[3] Georgia Institute of Technology, [4] National University of Singapore,
[5] Zhejiang University, [6] Renmin University of China,
[7] the Chinese University of Hong Kong
wt0318@connect.hku.hk

## Abstract

Large Language Models (LLMs) have demonstrated strong mathematical reasoning abilities through supervised fine-tuning and reinforcement learning. However, existing Process Reward Models (PRMs) are vulnerable to reward hacking and require expensive, large-scale annotation of reasoning steps, limiting their reliability and scalability. To address the first problem, we propose a novel reward model approach, **H**ierarchical **R**eward **M**odel **(HRM)**, which evaluates both individual and consecutive reasoning steps from fine-grained and coarse-grained level. HRM excels at assessing multi-step mathematical reasoning coherence, particularly in cases where a flawed step is later corrected through self-reflection. Furthermore, to address the inefficiency of autonomously annotating PRM training data via Monte Carlo Tree Search (MCTS), we propose a lightweight data augmentation strategy, **H**ierarchical **N**ode **C**ompression (**HNC**), which merges consecutive reasoning steps within the tree structure. Applying HNC to MCTS-generated reasoning trajectories increases the diversity and robustness of HRM training data, while introducing controlled noise with minimal computational overhead. Empirical results on the PRM800K dataset demonstrate that HRM, in conjunction with HNC, achieves superior stability and reliability in evaluation compared to PRM. Furthermore, cross-domain evaluations on MATH500 and GSM8K dataset confirm HRM's superior generalization and robustness across diverse mathematical reasoning tasks. The core code is publicly available [1].

## 1 Introduction

As the scale of parameters in LLMs continues to grow (Anil et al., 2023; Achiam et al., 2023; Grattafiori et al., 2024; Yang et al., 2024a), their general capabilities have significantly improved, surpassing human performance in various generative tasks such as text comprehension and data generation (Wang et al., 2024a). However, the upper bound and inherent limitations of LLMs in reasoning-intensive tasks—such as mathematical reasoning—remain an open question (Cobbe et al., 2021; Lightman et al., 2023; Uesato et al., 2022; Wang et al., 2023; Luo et al., 2024; Setlur et al., 2025; Wang et al., 2024c,b). Recent approaches, such as Chain-of-Thought (CoT) (Wei et al., 2022) and Tree-of-Thought (ToT) (Yao et al., 2023), have significantly enhanced reasoning performance. Despite these advancements, most CoT models lack mechanisms to detect and correct intermediate reasoning errors, resulting in continued propagation of mistakes throughout the reasoning process. Meanwhile, ToT methods do not inherently verify every intermediate step or guarantee retrieval of the optimal reasoning trajectory, which can limit its reliability in complex problem-solving scenarios.

To mitigate these limitations, recent efforts have focused on reward mechanisms that guide LLMs effectively. There are two primary approaches to enhance the reasoning capabilities of LLMs from the perspective of "how to reward LLMs": the Outcome Reward Model (ORM) (Lightman et al., 2023; Uesato et al., 2022; Guo et al., 2025; Shao et al., 2024) and the Process Reward Model (PRM) (Lightman et al., 2023; Uesato et al., 2022). Each comes with its own limitations. ORM suffers from delayed feedback and credit assignment issues, making it difficult to pinpoint which reasoning steps contribute to the final answer (Lightman et al., 2023; Uesato et al., 2022). PRM, in contrast, provides finer-grained supervision by evaluating reasoning step by step. However, most PRM methods are model-based and are prone to reward hacking (Weng, 2024), where models exploit reward signals rather than genuinely improving reasoning, undermining reliability in complex tasks. More-

*Corresponding author

[1] https://github.com/tengwang0318/hierarchial_reward_model



*Findings of the Association for Computational Linguistics: ACL 2026*, pages 565–576
July 2-7, 2026 

| Feature | ORM | PRM | HRM |
|---|---|---|---|
| **Scoring Method** | Rule-Based or RM | RM Only | RM Only |
| **Granularity (Training Data)** | Whole Process | Single Step | Few Consecutive Steps |
| **Step-wise Feedback** | No | Yes | Yes |
| **Error Correction** | Yes | No | Yes |

Table 1: Comparison of scoring methods, granularity, and feedback mechanisms across ORM, PRM, and HRM.

over, the high annotation cost associated with PRM makes large-scale deployment challenging.

In this paper, we focus on addressing the limitations of PRM. To mitigate the impact of reward hacking in PRM, we propose the **H**ierarchical **R**eward **M**odel (**HRM**). The term **hierarchical** highlights that, during training, HRM incorporates a hierarchical supervision signal by evaluating reasoning processes at both fine-grained (single-step) and coarse-grained (consecutive multi-step) levels. This layered approach enables HRM to capture both local and global coherence in reasoning. However, during inference, HRM remains step-wise: it assigns rewards to each reasoning step individually, the same as PRM. Traditional PRM penalizes an incorrect single step without considering potential corrections in subsequent reasoning. In contrast, HRM assesses reasoning coherence across multiple steps, allowing the reward model to identify and incorporate later steps that rectify earlier errors, leading to a more robust and reliable evaluation. Table 1 compares the difference between ORM, PRM and HRM.

While HRM can be applied to other structured reasoning domains, we focus on *mathematical reasoning*, the only domain with large-scale human-annotated process-level data for supervised reward modeling. The PRM800K (Lightman et al., 2023) dataset comprises manually annotated reasoning trajectories, which serve as the foundation for training ORM, PRM, and HRM. We subsequently assess the performance of Qwen2.5-72B-Math-Instruct (Yang et al., 2024b) as the policy model by employing the Best-of-N Search strategy across ORM, PRM, and HRM. Experimental results (as shown in Table 2) demonstrate that HRM yields the most stable and reliable reward evaluations. The policy model with HRM maintains stable performance, with accuracy stabilizing at 80% as $N$ increases. In contrast, policy models with PRM and ORM exhibit significant performance fluctuations, with accuracy degrading as $N$ grows.

To fully exploit the capabilities of Monte Carlo Tree Search (MCTS) for automatic process annotation, we introduce a data augmentation framework termed **H**ierarchical **N**ode **C**ompression (**HNC**), which consolidates two consecutive nodes from different depths into a single node. This approach effectively expands the training dataset while maintaining minimal computational overhead and enhancing label robustness through controlled noise injection. After evaluating HNC in the auto-annotation process by MCTS on the PRM800K dataset, we find that fine-tuned HRM achieves more robust scoring within PRM800K dataset and exhibits strong generalization across other domains, including GSM8K (Cobbe et al., 2021) and MATH500 (Lightman et al., 2023) dataset, outperforming PRM in robustness and consistency.

Our main contributions are as follows:

- We propose the **HRM**, which leverages hierarchical supervision from training data at both single-step and multi-step levels, promoting coherence and self-correction in multi-step reasoning. We validate HRM's robustness on the PRM800K dataset using manually annotated data.

- We introduce **HNC**, a lightweight data augmentation approach for MCTS that substantially increases the diversity and robustness of HRM training data with minimal computational cost. Experiments show that HRM trained on the PRM800K dataset with auto-annotated data from HNC and MCTS demonstrates improved robustness over PRM. Furthermore, HRM exhibits superior reasoning consistency and generalization across GSM8K and MATH500, consistently outperforming PRM.

- Additionally, we enhance the policy model through fine-tuning on high-quality reasoning trajectories filtered from MCTS, further improving its reasoning performance.

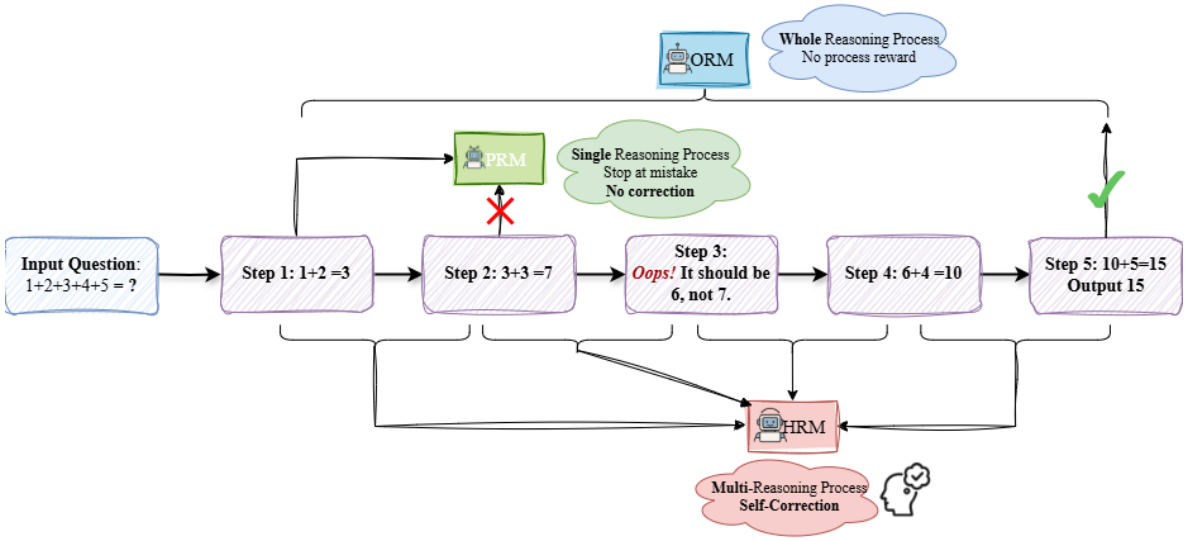



Figure 1: Illustration of how ORM, PRM, and HRM handle reasoning processes. ORM evaluates the entire reasoning chain, PRM assesses individual steps but stops at errors, and HRM considers multiple consecutive steps, enabling error correction. The figure also demonstrates how HRM constructs its training dataset by merging two consecutive steps.

## 2 Related Work

### 2.1 RLHF

Reinforcement Learning with Human Feedback (RLHF) (Ouyang et al., 2022) is a widely used framework for optimizing LLMs by incorporating human feedback. The core idea of RLHF is to use an RM to distinguish between high-quality and low-quality responses and optimize the LLM using PPO (Schulman et al., 2017).

From the perspective of reward design, there are two main approaches: ORM (Cobbe et al., 2021; Lightman et al., 2023; Wang et al., 2023) and PRM (Lightman et al., 2023; Uesato et al., 2022; Luo et al., 2024; Zhang et al., 2024). ORM assigns rewards based on the whole output, while PRM evaluates intermediate reasoning steps to provide more fine-grained supervision.

### 2.2 ORM

ORM suffers from delayed feedback and the credit assignment problem. Since rewards are only provided at the final outcome, ORM struggles to discern which intermediate steps contribute to success or failure (Lightman et al., 2023). This delayed feedback limits learning efficiency, making it harder to optimize critical decision points. Additionally, ORM is prone to spurious reasoning (Cobbe et al., 2021; Wang et al., 2023), where the model arrives at the correct answer despite flawed intermediate steps, reinforcing suboptimal reasoning patterns. However, DeepSeek-R1(Guo et al., 2025) integrates a rule-based ORM within GRPO(Shao et al., 2024), demonstrating that rule-based reward, rather than score-based reward models, can effectively guide LLMs toward generating long-CoT reasoning and self-reflection, ultimately enhancing their reasoning abilities.

### 2.3 PRM

One of the most critical challenges in PRM is reward hacking, a phenomenon in which an RL agent exploits flaws or ambiguities in the reward function to achieve artificially high rewards without genuinely learning the intended task or completing it as expected (Amodei et al., 2016; Di Langosco et al., 2022; Everitt et al., 2017; Weng, 2024).

Furthermore, the annotation process required for training PRM is prohibitively expensive (Lightman et al., 2023; Uesato et al., 2022), making large-scale implementation impractical. To address this issue, MCTS has been proposed as an autonomous method for annotating reasoning trajectories (Luo et al., 2024; Zhang et al., 2024). Moreover, in MCTS, the computational cost increases significantly as the depth and breadth of the search tree expand. To mitigate this, constraints are imposed on both tree height and width, limiting the number of simulation steps and thereby reducing the diversity of generated reasoning data.

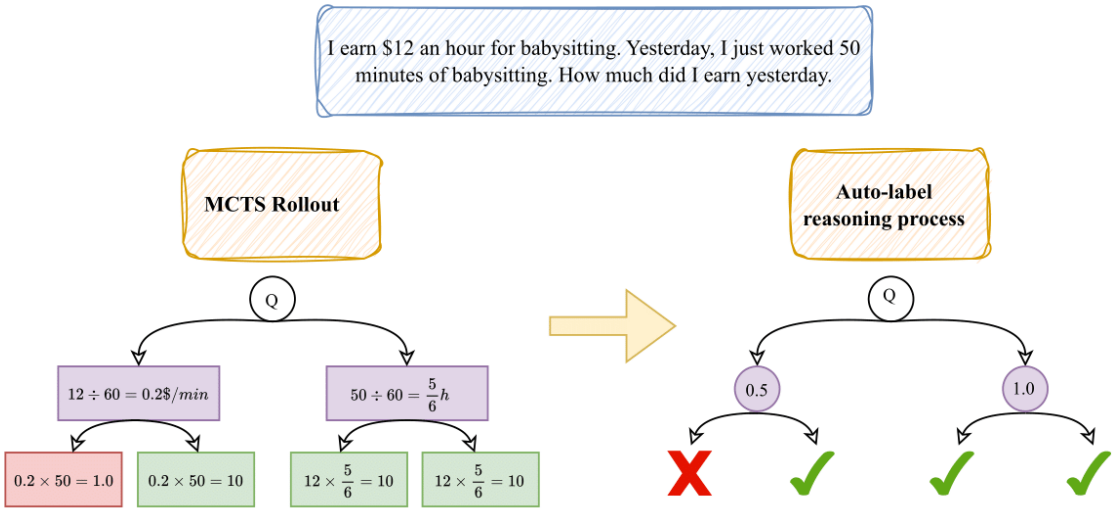



Figure 2: Illustration of the MCTS-based automated reasoning annotation process. The left side depicts a tree structure where each node represents a reasoning step, simulated using the ToT approach with MCTS. The right side visualizes the assigned scores for each step in the reasoning tree.

## 3 Methodology

### 3.1 Hierarchical Reward Model

PRM provides fine-grained, step-wise supervision, whereas ORM evaluates reasoning holistically. To combine the advantages of both, we propose the **H**ierarchical **R**eward **M**odel (**HRM**), which introduces hierarchical supervision by training on both single-step and consecutive multi-step reasoning sequences. HRM serves as an evaluation-oriented reward model that estimates the relative quality of reasoning trajectories, rather than directly optimizing the policy model itself. This hierarchical design enables HRM to capture both local accuracy and global coherence, yielding more stable and reliable reward evaluation in multi-step reasoning. The HRM training data extends PRM supervision by merging consecutive reasoning steps into hierarchical segments (from step 1 to $N$), as illustrated in Fig. 1, forming a superset of the PRM dataset.

Formally, let $D$ represent the training dataset, $N$ denote the total number of reasoning steps in a sequence, $s_i$ be the $i$-th reasoning step, and $R(\cdot)$ be the reward function that assigns a score to a step. The training datasets for PRM and HRM are defined as:

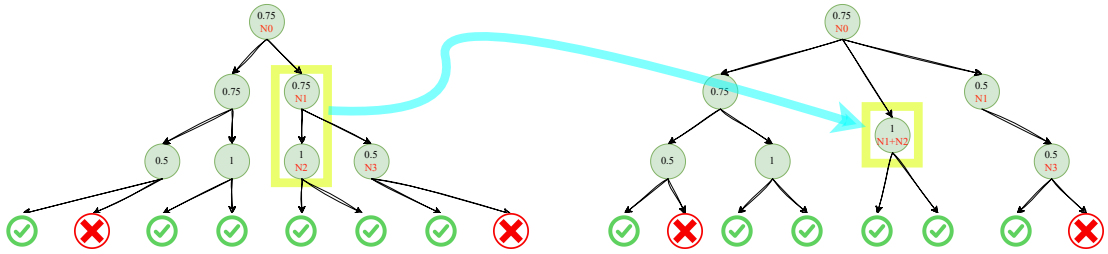
$$D_{\text{PRM}} = \{(s_i,\, R(s_i)) \mid 1 \le i \le N\}\,, \tag{1}$$

$$D_{\text{HRM}} = D_{\text{PRM}} \cup \{(s_i + s_{i+1},\, R(s_i + s_{i+1})) \mid 1 \le i < N\}\,. \tag{2}$$

Hierarchical supervision aims to connect local correctness with global coherence by jointly evaluating individual steps and their interactions. In practice, HRM achieves this through two objectives: (1) capturing both fine-grained and coarse-grained consistency, and (2) enabling self-reflection and error correction. Unlike PRM, which evaluates each step independently and penalizes early mistakes, HRM considers whether later steps can repair earlier errors, treating them as a coherent reasoning process rather than isolated faults.

Although HRM training data incorporates merged reasoning steps, the model remains step-wise in inference, assigning a reward based solely on the current step $s_i$, similar to PRM.

This design preserves fine-grained step-level evaluation while mitigating the subjectivity of step segmentation: by merging neighboring steps during training, HRM smooths annotation noise from ambiguous boundaries and enhances robustness across reasoning granularities. Conceptually, this hierarchical supervision extends PRM in a principled way, integrating both local accuracy and global coherence. In this work, we instantiate HRM




**Step 1:** Merge the reasoning steps of **N1** and **N2**, integrating their information into **N2**.
**Step 2:** Remove the direct connection between **N1** and **N2**.
**Step 3:** Redirect **N2** to point to **N0**.

Figure 3: Illustration of HNC. The left part represents the original MCTS data annotation structure, while the right part shows the transformed MCTS structure after applying HNC.

with two-step merging as the minimal yet effective form of hierarchical supervision, enabling controlled analysis while maintaining computational efficiency.

### 3.2 Hierarchical Node Compression in MCTS

Due to the prohibitively high cost of human-annotated supervision, autonomous annotation methods based on MCTS have been proposed. Fig. 2 illustrates the process of automatic reasoning annotation using MCTS. Given a ground truth and a corresponding question, MCTS generates multiple possible reasoning paths by simulating different step-by-step solutions. Each node in the search tree represents a reasoning step, and its score is calculated based on the proportion of correct trajectories in its subtree, reflecting the likelihood that the reasoning path is valid. However, these methods demand substantial computational resources, as achieving reliable estimates of intermediate reasoning step scores requires a sufficiently deep and wide search tree to reduce variance and mitigate bias; otherwise, the estimates may remain unreliable. This exponential growth in complexity makes large-scale implementation challenging.

To better leverage autonomous process annotation, we propose a data augmentation method called **H**ierarchical **N**ode **C**ompression (**HNC**). The key idea is to merge two consecutive nodes, each corresponding to a reasoning step, into a single node, thereby creating a new branch with minimal computational overhead.

As shown in Fig. 3, HNC assumes that each node has a sufficiently large number of child nodes. By randomly merging consecutive nodes, it introduces controlled noise, enhancing the robustness of MCTS-based scoring. Before HNC, each child node contributes $\frac{1}{N}$ to the total score. After HNC removes a random node, the remaining child nodes redistribute their weights to $\frac{1}{N-1}$, increasing their individual influence. Since child nodes are independent and identically distributed from the parent's perspective, the expectation of the parent score remains unchanged. However, the variance increases from $\frac{\sigma^2}{N}$ to $\frac{\sigma^2}{N-1}$, introducing controlled noise that enables data augmentation at an extremely low computational cost. When $N$ is sufficiently large, this variance change remains moderate while still facilitating effective data augmentation.

### 3.3 Self-Training

We filter high-quality reasoning data from MCTS using the MC-Score to ensure reliable supervision and avoid reward bias. Due to computational constraints, we do not employ RL methods such as PPO (Schulman et al., 2017) or GRPO (Shao et al., 2024). Instead, we continue using supervised fine-tuning. To preserve the general capabilities of the policy model, we incorporate causal language modeling loss combined with KL divergence regularization using a reference model. The objective function is defined as:

$$\mathcal{L} = \mathcal{L}_{\text{LM}} + \lambda \log D_{\text{KL}}(P||Q), \tag{3}$$

where $\mathcal{L}_{\text{LM}}$ represents the causal language modeling loss computed on high-quality reasoning sequences, and $D_{\text{KL}}(P||Q)$ denotes the KL divergence between the policy model's output distribution $P$ and the reference model's output distribution $Q$. The term $\lambda$ serves as a weighting factor to balance task-specific adaptation and retention of general capabilities.

Without proper KL regularization or with an insufficiently weighted KL loss (i.e., a very small $\lambda$), the KL divergence grows unbounded during training. Specifically, KL loss typically ranges from 0 to

| N | 2 | 4 | 8 | 16 | 24 |
|---|---|---|---|---|---|
| **ORM** | 0.622 | 0.677 | 0.655 | 0.655 | 0.633 |
| **PRM** | 0.700 | 0.644 | 0.611 | 0.588 | 0.577 |
| **HRM** | **0.722** | **0.711** | **0.744** | **0.800** | **0.800** |

Table 2: Accuracy of Qwen2.5-72B-Math-Instruct on the PRM800K test set under the best-of-$N$ strategy, comparing ORM, PRM, and HRM. All models are fine-tuned on the manually labeled PRM800K dataset.

20000, whereas the causal LM loss remains within 0 to 12, leading to a severe loss imbalance. This causes the optimization process to excessively minimize KL divergence at the expense of task-specific reasoning performance. To address this, we apply a logarithmic scaling to $D_{\text{KL}}(P||Q)$, stabilizing the loss landscape and ensuring a balanced trade-off between preserving general language capabilities and enhancing reasoning ability. Further details are provided in Section 4.3.

## 4 Experiment

### 4.1 HRM

Given that the PRM800K dataset consists of Phase1 and Phase2, where Phase1 includes manually annotated reasoning processes, we utilize these manual annotations to construct the training datasets for ORM, PRM, and HRM. ORM training data comprises complete reasoning trajectories, while PRM training data consists of individual reasoning steps conditioned on preceding context. HRM training data extends PRM by incorporating multiple consecutive reasoning steps, allowing HRM to capture self-reflection and ensure reasoning coherence across sequential steps. Table 6 summarizes the labeling rules for merged reasoning steps in HRM.

We fine-tune Qwen2.5-1.5B-Math (Yang et al., 2024b) as the RM for classifying given reasoning step as correct or incorrect. Given an input, RM predicts logits for the *positive* and *negative* classes, denoted as $l_{\text{pos}}$ and $l_{\text{neg}}$, respectively. The score is obtained by applying the softmax function:

$$P(y = \text{pos} \mid x) = \frac{\exp(l_{\text{pos}})}{\exp(l_{\text{pos}}) + \exp(l_{\text{neg}})}, \quad (4)$$

where $P(y = \text{pos} \mid x)$ denotes the probability of given reasoning step being correct. This probability serves as the reward assigned by RM. Detailed information is provided in Appendix A.2.

To evaluate the performance of ORM, PRM, and HRM, we employ Qwen2.5-72B-Math-Instruct (Yang et al., 2024b) as the policy model and implement the best-of-$N$ strategy. Specifically, ORM selects the best result from $N$ complete reasoning trajectories, while PRM and HRM score $N$ intermediate reasoning steps and select the most promising one at each step. For PRM and HRM, we consider the completion of a formula as an intermediate reasoning step, enabling a finer-grained evaluation mechanism. Table 2 presents the results, showing that the accuracy of the policy model with ORM and PRM exhibits significant fluctuations, decreasing as $N$ increases. In contrast, the policy model with HRM maintains stable performance, converging to an accuracy of 80% as $N$ grows.

**Statistical robustness.** The evaluation itself already provides extensive statistical averaging. Each dataset contains thousands of test problems (1k in GSM8K, 500 in MATH500, and over 800k annotated reasoning steps in PRM800K), which smooths random variation across samples. Notably, MATH500 and PRM800K consist of challenging high-school and university-level problems (e.g., calculus, linear algebra), whose solutions require multi-step reasoning rather than surface-level pattern matching or guessing. Moreover, in the Best-of-$N$ evaluation (up to $N = 512$), each reasoning step expands into $N$ candidate paths, forming an exponentially large search tree. This hierarchical exploration implicitly averages over a vast number of trajectories, effectively functioning as multi-seed sampling and significantly reducing evaluation variance.

### 4.2 HNC

In this section, we utilize only the questions and ground truth from the PRM800K dataset, without relying on manually annotated data. We adopt MCTS as the automatic annotation method, using Qwen2.5-7B-Math-Instruct as the reasoning engine to generate trajectories. As mentioned in Section 3.2, these auto-annotated reasoning trajectories from MCTS are used to train PRM, after which we apply the HNC data augmentation method to generate additional training data for HRM.

To balance computational efficiency and robustness, we configure MCTS with 5–6 child nodes per parent and a maximum tree depth of 7, ensuring reasoning completion within 7 steps. Since the computational cost of MCTS rollouts grows exponentially with tree depth and branching factor, we limit these parameters to maintain feasibility.

| Policy Model | Method | N | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 2 | 4 | 8 | 16 | 24 | 32 | 64 | 128 | 256 | 512 |
| DeepSeek-Math-7B | PRM | 0.311 | 0.433 | 0.377 | 0.455 | 0.411 | 0.455 | 0.466 | 0.444 | 0.377 | 0.377 |
| | HRM | 0.311 | 0.388 | **0.444** | 0.455 | **0.455** | 0.422 | **0.533** | **0.522** | **0.455** | **0.500** |
| Qwen2.5-72B-Math | PRM | 0.233 | 0.344 | 0.411 | 0.422 | 0.488 | 0.522 | 0.600 | 0.566 | 0.666 | 0.700 |
| | HRM | **0.288** | **0.366** | 0.366 | **0.488** | **0.511** | **0.611** | **0.622** | **0.611** | **0.711** | **0.722** |
| Qwen2.5-7B-Math | PRM | 0.477 | 0.466 | 0.600 | 0.544 | 0.633 | 0.677 | 0.733 | 0.677 | 0.700 | 0.722 |
| | HRM | **0.500** | **0.566** | **0.655** | **0.600** | **0.666** | **0.711** | 0.711 | **0.766** | **0.777** | **0.766** |

Table 3: Accuracy of different policy models under PRM and HRM using the best-of-$N$ strategy on the PRM800K test set. The training data for both PRM and HRM are derived from MCTS with Qwen2.5-7B-Math-Instruct.

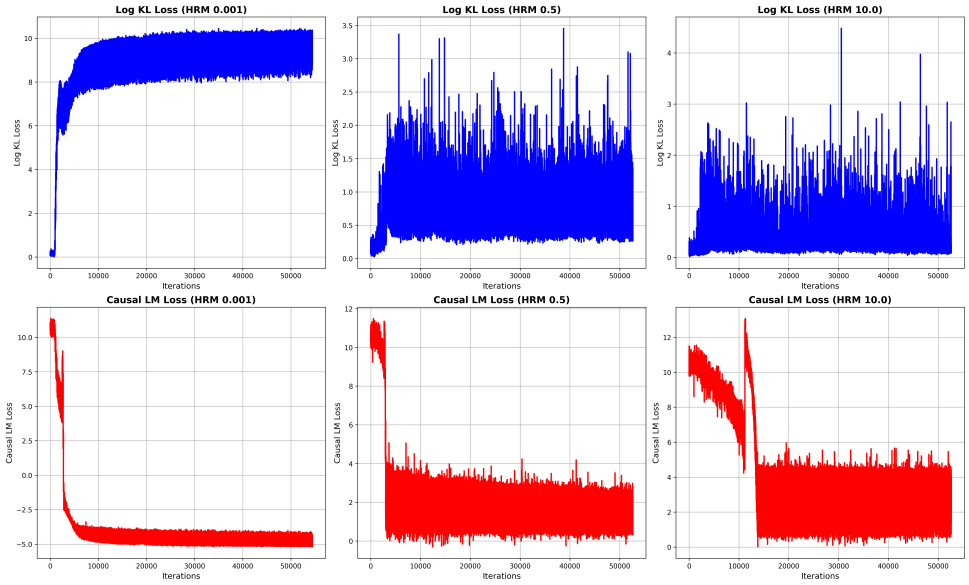



Figure 4: Loss dynamics during training across different KL loss weightings. Each column corresponds to a different $\lambda$ value: 0.001 (left), 0.5 (middle), and 10.0 (right). The top row shows the log KL loss, while the bottom row depicts the causal language modeling loss.

The full MCTS simulation requires approximately 2,457 A100 GPU-hours, while the HNC augmentation process takes around 30 minutes with CPU.

We perform supervised fine-tuning of Qwen2.5-1.5B-Math for both PRM and HRM. Training HRM adds roughly 30% more samples compared to PRM, increasing total training time from about 84 GPU hours to 120 GPU hours on A100 GPUs. This additional cost is negligible relative to the over 2400 GPU hours required for generating the reasoning trajectories via MCTS, which constitute the true computational bottleneck.

To evaluate performance, we employ different policy models, including Qwen2.5-7B-Math-Instruct (Yang et al., 2024b), DeepSeek-Math-7B (Shao et al., 2024), and Qwen2.5-72B-Math-Instruct (Yang et al., 2024b), applying the best-of-$N$ strategy on the PRM800K dataset. Detailed training information is provided in Appendix A.3.

Table 3 presents the accuracy results of various policy models under PRM and HRM settings on the PRM800K dataset. Although both PRM and HRM training data are derived from MCTS with Qwen2.5-7B-Math-Instruct, we evaluate HRM and PRM using different policy models, where HRM consistently demonstrates greater stability and robustness compared to PRM.

The relatively lower performance of Qwen2.5-72B-Math-Instruct (as shown in Table 3) can be attributed to the tree height constraints imposed by MCTS, which limit answer generation to a predefined template of at most 6 reasoning steps and require explicit output of "# Step X". While Qwen2.5-72B-Math-Instruct exhibits strong reasoning capabilities, its highly structured training process makes it more likely to generate outputs

| Policy Model | Method | N | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 2 | 4 | 8 | 16 | 24 | 32 | 64 | 128 | 256 | 512 |
| Qwen2.5-7B-Math | PRM | 0.477 | 0.466 | 0.600 | 0.544 | 0.633 | 0.677 | 0.733 | 0.677 | 0.700 | 0.722 |
| | HRM | **0.500** | **0.566** | **0.655** | **0.600** | **0.666** | **0.711** | 0.711 | **0.766** | **0.777** | **0.766** |
| Qwen-7B-PRM | PRM | 0.477 | 0.555 | 0.533 | 0.655 | 0.655 | 0.677 | 0.711 | 0.700 | 0.744 | 0.744 |
| | HRM | 0.477 | 0.544 | **0.644** | **0.722** | **0.700** | **0.711** | **0.722** | **0.755** | **0.800** | **0.800** |
| Qwen-7B-HRM | PRM | 0.511 | 0.533 | 0.644 | 0.667 | 0.667 | 0.689 | 0.733 | 0.722 | 0.744 | 0.733 |
| | HRM | 0.489 | **0.589** | **0.722** | **0.722** | **0.722** | **0.744** | **0.744** | **0.789** | **0.789** | **0.789** |

Table 4: Comparison of fine-tuned policy model reasoning performance on the PRM800K dataset using the best-of-$N$ strategy. Qwen-7B-HRM denotes the policy model fine-tuned on high-MC-score reasoning data from HRM's training set, while Qwen-7B-PRM follows the same procedure for PRM's training set.

| Dataset | Method | N | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 2 | 4 | 8 | 16 | 24 | 32 | 64 | 128 | 256 | 512 |
| GSM8K | PRM | 0.784 | 0.828 | 0.858 | 0.882 | 0.884 | 0.893 | 0.905 | 0.917 | 0.927 | 0.918 |
| | HRM | 0.784 | **0.833** | 0.846 | **0.886** | **0.893** | **0.902** | **0.907** | 0.914 | **0.930** | **0.926** |
| MATH500 | PRM | 0.468 | 0.572 | 0.598 | 0.624 | 0.658 | 0.658 | 0.656 | 0.662 | 0.686 | 0.688 |
| | HRM | **0.490** | **0.576** | **0.612** | **0.660** | **0.688** | **0.692** | **0.742** | **0.740** | **0.740** | **0.736** |

Table 5: Generalization performance of PRM and HRM trained on PRM800K and evaluated on GSM8K and MATH500 under the Best-of-$N$ setting using Qwen2.5-7B-Math-Instruct.

that deviate from the required format. As a result, some outputs cannot be retrieved using regex-based post-processing, thereby affecting the overall measured performance.

### 4.3 Self-Training

We adapt the method described in Section 3.3 to filter high-quality reasoning data and train the policy model. Fig. 4 illustrates that when $\lambda$ is small (e.g., 0.001), the fine-tuned policy model rapidly loses its generalization ability within just a few iterations, causing the KL loss to escalate to approximately 20,000. In contrast, the causal LM loss remains within the range of 0 to 12, leading to a significant imbalance. This discrepancy underscores the necessity of applying logarithmic scaling to the KL term in the objective function, as discussed in Section 3.3. Conversely, when $\lambda$ is excessively large (e.g., 10.0), the model prioritizes adherence to the reference distribution, resulting in slower convergence and constrained improvements in reasoning capability.

We fine-tune Qwen2.5-7B-Math-Instruct using reasoning data with MC-score $> 0.9$, extracted from the PRM and HRM training datasets. Qwen-7B-HRM denotes the policy model fine-tuned on such data from HRM's training set, while Qwen-7B-PRM follows the same procedure for PRM's training set. We set $\lambda$ to 0.5. Table 4 further validates that SFT enhances the policy model's reasoning capability by leveraging high-quality data, with HRM demonstrating greater robustness compared to PRM.

### 4.4 HRM Generalization Across Different Domains

Trained solely on PRM800K, HRM and PRM are further evaluated on MATH500 (Lightman et al., 2023) and GSM8K (Cobbe et al., 2021) to assess their generalization to out-of-domain reasoning tasks. Table 5 shows that HRM exhibits greater robustness across different domains, demonstrating superior generalization performance, particularly in MATH500, where it effectively handles complex mathematical reasoning tasks.

However, the performance difference between HRM and PRM on the GSM8K dataset is marginal, as GSM8K primarily consists of relatively simple arithmetic problems. A strong policy model can typically solve these problems within three steps, reducing the impact of HRM's key advantages, such as assessing multi-step reasoning coherence and facilitating self-reflection. Nevertheless, as shown in Table 5, HRM still achieves a slight per-

formance edge over PRM, even on simpler datasets like GSM8K.

## 5 Conclusion

In this paper, we present HRM, which enhances multi-step reasoning evaluation by integrating fine-grained and coarse-grained assessments, improving reasoning coherence and self-reflection. We further introduce HNC, a data augmentation method that optimizes MCTS-based autonomous annotation, enhancing label diversity while expanding training data with minimal computational cost. Extensive experiments on PRM800K dataset demonstrate HRM's superior robustness over PRM, with strong generalization across GSM8K and MATH500 dataset. Additionally, MCTS-generated auto-labeled data enables policy model fine-tuning, further improving reasoning performance.

## 6 Limitations

### 6.1 Merged Steps

The reason why we only merge two consecutive steps at a time is to maintain the simplicity and clarity of the labeling strategy. When merging more than two steps—such as combining four reasoning steps—the number of possible label combinations increases rapidly (e.g., one positive, two negative, one neutral), making it difficult to define unified labeling rules and leading to potential conflicts. This stands in sharp contrast to the straightforward rules shown in Table 6, where merging only two steps allows for clear and consistent label definitions.

## References


Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. In *ArXiv e-prints*.

Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, and 1 others. 2022. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15. IEEE.

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. 2016. Concrete problems in ai safety. In *arXiv preprint arXiv:1606.06565*.

Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, and 1 others. 2023. Palm 2 technical report. In *ArXiv e-prints*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. In *ArXiv e-prints*.

Tri Dao. 2024. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Lauro Langosco Di Langosco, Jack Koch, Lee D Sharkey, Jacob Pfau, and David Krueger. 2022. Goal misgeneralization in deep reinforcement learning. In *International Conference on Machine Learning*, pages 12004–12019. ICML.

Tom Everitt, Victoria Krakovna, Laurent Orseau, Marcus Hutter, and Shane Legg. 2017. Reinforcement learning with a corrupted reward channel. In *IJCAI*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. In *ArXiv e-prints*.

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

Dhiraj Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, Dipankar Das, Kunal Banerjee, Sasikanth Avancha, Dharma Teja Vooturi, Nataraj Jammalamadaka, Jianyu Huang, Hector Yuen, and 1 others. 2019. A study of bfloat16 for deep learning training. *arXiv preprint arXiv:1905.12322*.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. In *ICLR*.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, Jiao Sun, and 1 others. 2024. Improve mathematical reasoning in language models by automated process supervision. In *ArXiv e-prints*, volume 2.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and 1 others. 2017. Mixed precision training. *arXiv preprint arXiv:1710.03740*.

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

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE.

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

Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. 2025. Rewarding progress: Scaling automated process verifiers for llm reasoning. In *ICLR*.

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

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

Peiyi Wang, Lei Li, Zhihong Shao, RX Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2023. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *ACL*.

Teng Wang, Zhenqi He, Wing-Yin Yu, Xiaojin Fu, and Xiongwei Han. 2024a. Large language models are good multi-lingual learners: When llms meet cross-lingual prompts. In *COLING*.

Teng Wang, Wing-Yin Yu, Zhenqi He, Zehua Liu, Hailei Gong, Han Wu, Xiongwei Han, Wei Shi, Ruifeng She, Fangzhou Zhu, and 1 others. 2024b. Bpp-search: Enhancing tree of thought reasoning for mathematical modeling problem solving. *arXiv preprint arXiv:2411.17404*.

Teng Wang, Wing-Yin Yu, Ruifeng She, Wenhan Yang, Taijie Chen, and Jianping Zhang. 2024c. Leveraging large language models for solving rare mip challenges. *arXiv preprint arXiv:2409.04464*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Lilian Weng. 2024. Reward hacking in reinforcement learning.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024a. Qwen2. 5 technical report. In *ArXiv e-prints*.

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

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Deliberate problem solving with large language models. In *NeurIPS*.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. 2024. Rest-mcts*: Llm self-training via process reward guided tree search. *Advances in Neural Information Processing Systems*, 37:64735–64772.

# A Appendix

## A.1 Labeling Rules for HRM Training Data

Table 6 summarizes the labeling rules for merging reasoning steps in HRM, as applied to the manually labeled PRM800K dataset. We note that labeling consecutive neutral steps as negative is a deliberate design choice, motivated by the observation that such sequences often indicate non-progressive reasoning and inefficiency, which we aim to penalize to encourage concise and goal-directed reasoning.

## A.2 HRM Training Details

To accelerate the training process of reward model, we employ FlashAttention (Dao et al., 2022; Dao, 2024), DeepSpeed (Rajbhandari et al., 2020; Aminabadi et al., 2022), and mixed-precision training (Kalamkar et al., 2019; Micikevicius et al., 2017). However, within the PRM800K domain, we frequently encounter the issue: *"Current loss scale already at minimum - cannot decrease scale anymore. Exiting run."* This indicates that the numerical precision is insufficient for stable training. To mitigate this issue and ensure reproducibility, we set `max_grad_norm` to 0.01, which effectively stabilizes the training process.

We define the completion of a reasoning step as the end of a formula, using `stop = ['\]\n\n', '\)\n\n', '# END!']` as boundary markers.

The following prompt is used in Section 4.1:

```
"""
You are an expert of Math and need to
solve the following question and return
the answer.

Question:
{question}

Let's analyze this step by step.

After you finish thinking, you need to
output the answer again!

The answer should start with '# Answer',
followed by two line breaks and the
final response.

Just provide the answer value without
any descriptive text at the end.

And the answer ends with '# END!'
Below is a correct example of the
expected output format:
-----------------
Question: 1+2+3 = ?

Firstly, solve 1 + 2 = 3,
Then, 3 + 3 = 6.

# Answer

6
# END!
-----------------
"""
```

## A.3 HNC Setting Details

To ensure the feasibility of autonomous annotation using MCTS, we impose constraints on both the width and height of the search tree. This limitation prevents us from treating the completion of a formula as a single reasoning step. Instead, we require the model to explicitly output `# Step X` at each step. Consequently, the training data for the reward model is segmented using `# Step X` as a delimiter. During inference, we also apply `# Step X` as a separator and employ the Best-of-N strategy for selecting the optimal reasoning path.

The prompt we use is as follows(delimiter=`['# END!', '# Step 2', "# Step 3", "# Step 4", "# Step 5"]`):

```
"""You are an expert of Math and need to
solve the following question and return
the answer.

Question:
{question}


Let's analyze this step by step.

Begin each step with '# Step X' to
clearly indicate the entire reasoning
step.

After you finish thinking, you need to
output the answer again!

The answer should start with '# Answer',
followed by two line breaks and the
final response.

Just provide the answer value without
any descriptive text at the end.

And the answer ends with '# END!'

Below is a correct example of the
expected output format:
-----------------
Question: 1+2+3 = ?

# Step 1
solve 1 + 2 = 3,

# Step 2
Then, 3 + 3 = 6.

# Answer

6
```

| Previous Step Label | Current Step Label | **Label for Merged Step** |
|---|---|---|
| Positive | Positive | **Positive** |
| Positive | Neutral/Negative | **Negative** |
| Neutral/Negative | Positive | **Positive** |
| Neutral/Negative | Neutral/Negative | **Negative** |

Table 6: Labeling strategy for constructing the HRM training dataset from manual annotations in PRM800K dataset. PRM800K dataset contains three label types: Positive, Negative, and Neutral. HRM extends PRM by incorporating multi-step reasoning.

```
# END!
-----------------
"""
```

### A.4 Self-Training

Initially, KL loss is not incorporated, causing the policy model to lose its generalization ability rapidly, despite a continuous decrease in evaluation loss. To address this issue, we introduce KL loss to regularize training from the reference model.

The logarithmic scaling and weighting factor $\lambda$ are added to balance the impact of KL divergence. Without these adjustments, KL loss would range from 0 to 20000, while the language modeling loss remains between 0 and 12, leading to an imbalance. The logarithm ensures a more stable contribution of KL loss during training.

As illustrated in Fig. 4, setting $\lambda = 0.5$ achieves a balanced trade-off between KL loss and language modeling loss, preventing excessive divergence from the reference model while ensuring stable and effective training.