SCOPE: Compress Mathematical Reasoning Steps for Efficient Automated Process Annotation

Anonymous ACL submission

Abstract

Process Reward Models (PRMs) have demonstrated promising results in mathematical reasoning, but existing process annotation approaches, whether through human annotations or Monte Carlo simulations, remain computationally expensive. In this paper, we introduce Step COmpression for Process Estimation (SCOPE), a novel compression-based approach that significantly reduces annotation costs. We first translate natural language reasoning steps into code and normalize them through Abstract Syntax Tree, then merge equivalent steps to construct a prefix tree. Unlike simulation-based methods that waste numerous samples on estimation, SCOPE leverages a compression-based prefix tree where each root-to-leaf path serves as a training sample, reducing the complexity from $O(NMK)$ to $O(N)$. We construct a large-scale dataset containing 509K samples with only 5% of the computational resources required by previous methods. Empirical results demonstrate that PRMs trained on our dataset consistently outperform existing automated annotation approaches on both Best-of-N strategy and ProcessBench [1].

1 Introduction

As Large Language Models (LLMs) advance in complex reasoning tasks (Jaech et al., 2024; Liu et al., 2024; Yang et al., 2024; Dubey et al., 2024), designing effective reward models has become increasingly crucial. Process Reward Models (PRMs) (Uesato et al., 2022; Lightman et al., 2023) evaluate the reasoning process step-by-step, providing more fine-grained supervision than Outcome Reward Models (ORMs) that only assess final outputs (Cobbe et al., 2021; Yu et al., 2023). Recent studies consistently demonstrate PRMs' superior performance across complex reasoning tasks (Wang et al., 2024; Snell et al., 2024). But PRMs face a significant challenge: they require intensive human effort for data annotation, as every reasoning step needs a label (Lightman et al., 2023).

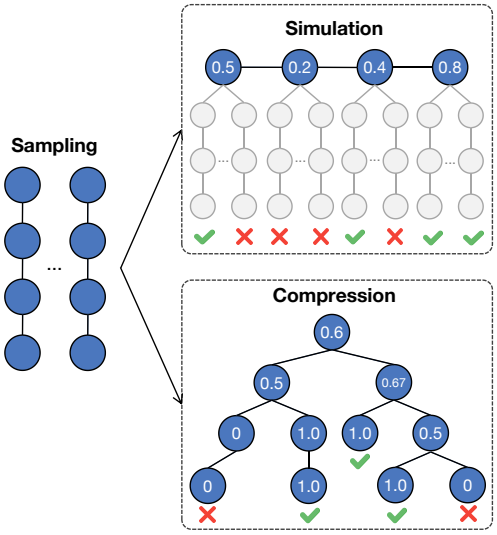

Figure 1: Comparison of PRM training data construction. Simulation-based methods require numerous completions solely for Q-value estimation, with these completions (gray nodes) being discarded without contributing to training. Our compression-based approach eliminates such data waste by merging equivalent steps from all sampled solutions into a prefix tree, where every root-to-leaf path becomes a valuable training instance.

To alleviate this limitation, recent studies explore simulation-based methods. For instance, Math-shephred (Wang et al., 2024) uses Monte Carlo estimation to automate the data annotation. As shown in Figure 1, it samples N solutions for a math problem, and then for each step in the solution, it simulates M potential completions and calculates the Q-value based on the proportion of completions that lead to the correct answer. While Math-shepherd eliminates human annotation requirements, it incurs high computational complexity of $O(NMK)$, where K denotes the average

[1] Our code, data, and models are available at https://github.com/anonymous.

step count of solutions. Recent studies have shown that Math-shepherd requires $38.8\times$ more FLOPs than ORM training (Yuan et al., 2024). Although OmegaPRM (Luo et al., 2024) reduces complexity to $O(NM \log K)$ through a divide-and-conquer strategy, the efficiency gains remain limited due to typically short lengths ($K < 10$). We argue that these simulation-based approaches are inherently inefficient as they generate numerous completions solely for Q-value estimation (gray area in the Figure 1), resulting in wasted data.

In this paper, we introduce **S**tep **CO**mpression for **P**rocess **E**stimation (**SCOPE**), a novel automatic PRM label annotation strategy that achieves $O(N)$ complexity while maintaining annotation quality. Unlike prior simulation-based methods (Wang et al., 2024; Luo et al., 2024), SCOPE introduces a novel compression-based approach: first sample numerous solutions for each problem, then merge equivalent solution steps to construct a prefix tree (Trie), as shown in Figure 1. For each node in the tree, its Q-value could be calculated as the proportion of solutions passing through it that reach the correct answer. Each path from root to leaf in the tree represents a training sample with step-by-step labels. Compared to simulation-based methods, SCOPE not only achieves $O(N)$ complexity through step compression, but also fully utilizes all sampled solutions by incorporating them directly into training data through the prefix tree structure.

The key challenge of SCOPE lies in identifying step equivalence. Naive exact string matching is too restrictive and results in limited compression. Edit distance and sentence embedding often fail to capture the subtle distinctions in mathematical reasoning (Wallace et al., 2019). To address this challenge, we propose a code-based step compression through a three-stage process: (1) translate natural language reasoning steps into executable Python code using a code LLM, (2) normalize the code through Abstract Syntax Tree (AST) (Aho et al., 2007) (e.g., variable renaming), and (3) merge steps with identical normalized code using a Trie. This approach enables precise identification of mathematically equivalent steps while being robust to surface-level variations. Although code translation and AST add computation, the overall complexity remains $O(N)$, ensuring substantially lower computational costs for large-scale PRM datasets.

Based on SCOPE, we construct a PRM training dataset containing 509K samples with 4M labels, exceeding Math-shepherd's scale while requiring only 5% of its computational resources. Empirical evaluation demonstrates the effectiveness of our approach, with PRMs trained on our dataset consistently outperforming other automated annotation approaches in both Best-of-N strategy and the ProcessBench (Zheng et al., 2024a). Our main contributions are:

- We propose SCOPE, a novel automatic PRM label annotation method that introduces a sample-and-compress paradigm to replace traditional sample-and-simulation paradigm, achieving $O(N)$ complexity.
- We introduce a new PRM training dataset containing 509K samples and 4M step labels with only 5% of previous MathShepherd's computational resources.
- Extensive experiments demonstrate that PRMs trained on our dataset achieve superior performance compared to other automated annotation approaches across multiple evaluation settings, including Best-of-N strategy and ProcessBench.

2 Related Work

2.1 PRMs Training

Process Reward Models (PRMs) demonstrate significant potential in mathematical reasoning tasks, though their traditional training approaches require substantial human annotation effort (Lightman et al., 2023). While Math-shepherd (Wang et al., 2024) introduces an innovative approach using Monte Carlo simulation to automate PRM training, its practical applications are constrained by intensive computational demands. OmegaPRM (Luo et al., 2024) attempts to address these limitations through a divide-and-conquer Monte Carlo Tree Search strategy, yet computational costs remain a significant barrier. Recent research explores alternative approaches to reduce these computational requirements: ImplicitPRM (Yuan et al., 2024) demonstrates the possibility of deriving PRMs from outcome-level labels, while AutoPSV (Lu et al., 2024) develops a novel verification model that evaluates step quality through confidence variation analysis. However, recent studies (Zheng et al., 2024a) have revealed that these approaches often fail short of their claimed effectiveness, particularly struggling on more challenging datasets.

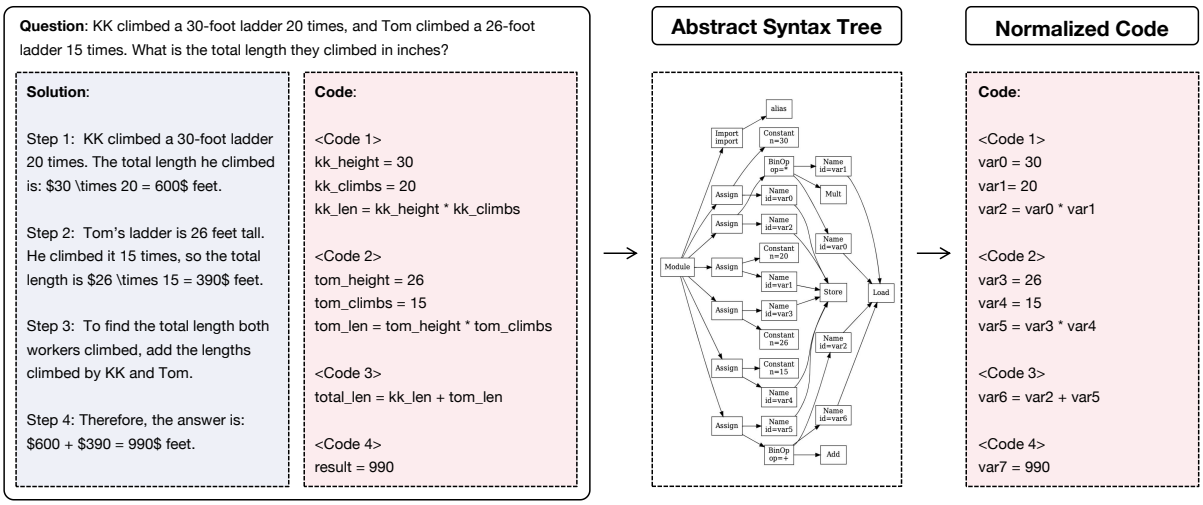

Figure 2: Illustration of code translation and normalization. The solution of a math problem is first converted into corresponding codes through a code-LLM. Then, we use AST module of Python to derive the abstract syntax tree. Finally, the codes are normalized via their corresponding AST.

2.2 PRMs in Mathematical Reasoning

Process Reward Models enhance mathematical reasoning capabilities through dual mechanisms: reinforcement learning during the training phase and solution selection during inference. In reinforcement learning (Wang et al., 2024; Yuan et al., 2024; Shao et al., 2024), PRMs serve as reward functions that guide policy optimization by providing fine-grained feedback on each reasoning step, enabling more targeted learning compared to traditional outcome-based rewards. During inference (Lu et al., 2024; Lightman et al., 2023; Wang et al., 2024), the effectiveness of PRMs is commonly evaluated using the Best-of-N strategy, which identifies the highest-quality solution from multiple candidates by aggregating step-wise scores, demonstrating superior performance compared to outcome-based selection methods. The recent introduction of ProcessBench (Zheng et al., 2024a) establishes a more rigorous framework for evaluating PRMs' capabilities in identifying erroneous reasoning steps, offering a comprehensive assessment of their process-level understanding. Building upon these insights into PRM effectiveness and evaluation frameworks, we comprehensively evaluated SCOPE on both Best-of-N strategy and ProcessBench. Our method not only achieved state-of-the-art performance on Best-of-N, but also demonstrated remarkable effectiveness on the challenging ProcessBench benchmark, outperforming the second-best method by 6.3%, validating the effectiveness of our compression-based approach.

3 Method

In this section, we present SCOPE, a novel approach for automatic PRM dataset annotation: (1) First, we motivate and explain our code translation strategy, which converts reasoning steps into executable code through a code LLM. (2) Then, we detail our step compression based on AST normalization to construct a prefix tree. (3) Finally, we describe our PRM training details.

3.1 Code Translation

As we have discussed in the Section 1, the key challenge of SCOPE lies in efficiently identifying and merging equivalent reasoning steps. A naive way using exact matching fails to recognize equivalent steps expressed differently (e.g., "multiply 5 by 3" vs "calculate 5×3"), leading to an overly sparse compression space. While edit distance or sentence embedding offer more flexibility, they struggle with precise numerical comparisons or operator precedence (e.g., failing to distinguish between "$(3 + 4) \times 2$" and "$3 + 4 \times 2$"), making them unreliable for mathematical scenario (Wallace et al., 2019). The core issue is that they operate on surface-level text similarities rather than identifying true mathematical equivalence, which can manifest in various forms such as different arithmetic representations or algebraic transformations.

Therefore, we propose using code as an intermediate representation that can precisely capture mathematical operations and logical reasoning. As shown in Figure 2, we first use the math LLM to

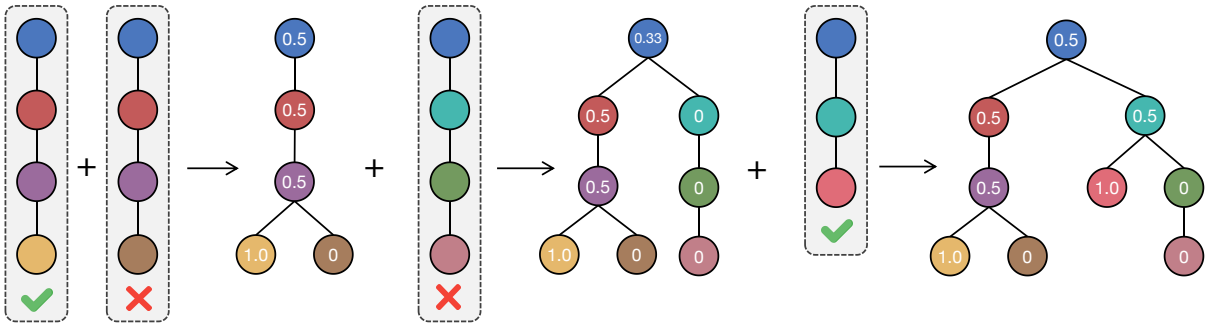

Figure 3: Visualization of prefix tree construction process. Same-colored nodes indicate equivalent normalized step codes. Q-values reflect the proportion of correct solutions passing through each node.

sample N solutions for each math problem, then employ a code LLM to convert each natural language step into executable code blocks (see prompt details in Appendix B). This code translation transforms natural language steps into a more structured and precise representation, laying the foundation for identifying mathematical equivalence. A detailed example of complex code generation is provided in Appendix C.

3.2 Step Compression

While code translation captures mathematical operations precisely, direct code matching remains ineffective due to syntactic variations. For example, "x = 5 * 3" and "result = 3 * 5" would be recognized as different operations despite being mathematically equivalent, due to different variable names and operand orders. To address this, we utilize Abstract Syntax Tree (AST), which represents code as a hierarchical structure of its syntactic elements, to normalize code through systematic transformations, as shown in Figure 2:

- Variable renaming: Mapping arbitrary variable names and function names to canonical form (e.g., var0, func0).
- Operation normalization: Standardizing equivalent operations (e.g., multiply/-times/product $\rightarrow$ mul).
- Expression reordering: Sorting commutative operations for consistent representation.
- Constant folding: calculating constant expressions (e.g., 2 * 3 $\rightarrow$ 6).

Through AST normalization, code blocks that are mathematically equivalent but syntactically different will have the same representation.

After AST normalization, we merge equivalent code blocks to construct a prefix tree (Trie), where each node represents a distinct solution step and edges denote reasoning branches. As illustrated in Figure 3, this tree structure naturally captures the shared reasoning patterns across different solutions. Since both AST normalization and Trie construction have linear complexity to the number of solutions, SCOPE maintains an overall complexity of $O(N)$. Moreover, this hierarchical representation enables efficient Q-value computation directly from the solution paths without requiring additional Monte Carlo simulations. The Q-value could calculated recursively by propagating the correctness of leaf solutions up through the tree, weighted by the number of solutions passing through each path. The code for Q-value calculation is shown below, and the pseudocode of the complete workflow of SCOPE is presented in Algorithm 1.

```
def compute_q_values(node):
  if node.is_leaf():
    return node.is_correct, node.count

  total_value, total_count = 0, 0
  for child in node.children:
    value, count = compute_q(child)
    total_value += value * count
    total_count += count

  q_value = total_value / total_count
  return q_value, total_count
```

3.3 PRM Training

Through the above step compression process, we obtain Q-values for each step, which naturally serve as labels for their corresponding reasoning steps. Following Math-shepherd, we explore two strategies to estimate the label y_{s_i} for the step s_i, hard estimation (HE) and soft estimation (SE). For HE, we assign binary labels based on the Q-value $Q(s_i)$ of step s_i: a positive Q-value indicates that at least

Algorithm 1 SCOPE: Automatic PRM Dataset Annotation with Step Compression

Require: Math problem P, Number of samples N, Math LLM M_{math}, Code LLM M_{code}
Ensure: Compressed solution space T with Q-values for PRM training

function SCOPE(P, N, M_{math}, M_{code})
  $S \leftarrow \emptyset$ ▷ Store all normalized solution paths
  for $i \leftarrow 1$ to N **do** ▷ Sample N solutions using Math LLM
    $steps \leftarrow M_{math}(P)$ ▷ Each step in natural language
    $code\_steps \leftarrow \{M_{code}(step) \mid step \in steps\}$ ▷ Convert all steps to executable code
    $norm\_steps \leftarrow \{\text{NormalizeAST}(code) \mid code \in code\_steps\}$ ▷ AST normalization
    $S \leftarrow S \cup norm\_steps$ ▷ Add normalized solution path to solution set
  end for
  $T \leftarrow \text{BuildPrefixTree}(S)$ ▷ Construct Trie from solution paths
  $Q \leftarrow \text{ComputeQValues}(T)$ ▷ Calculate Q-values by propagating correctness
  return (T, Q) ▷ Return compressed solution space
end function

one solution path through this step reaches the correct answer:

$$y_{s_i}^{HE} = \begin{cases} 1 & Q(s_i) > 0 \\ 0 & \text{Otherwise} \end{cases} \tag{1}$$

For SE, we directly use the Q-value as the label, which reflects the proportion of paths from this step that reach the correct answer:

$$y_{s_i}^{SE} = Q(s_i) \tag{2}$$

We adopt different loss functions for HE and SE to align with their respective label characteristics. For HE with binary labels, we use the binary cross-entropy loss for optimization:

$$\mathcal{L}_{\text{HE}} = -\sum_{i=1}^{K} y_{s_i} \log \hat{y}_{s_i} + (1 - y_{s_i}) \log(1 - \hat{y}_{s_i}) \tag{3}$$

where $\hat{y}_{si}$ is the model's predicted probability for step s_i, and K is the total number of steps. For SE with continuous Q-values as labels, we employ the mean squared error (MSE) loss:

$$\mathcal{L}_{\text{SE}} = -\sum_{i=1}^{K} (y_{s_i} - \hat{y}_{s_i})^2 \tag{4}$$

The choice of different loss functions reflects the distinct nature of HE and SE: binary cross-entropy is suited for classification tasks with hard labels, while MSE better handles regression with continuous values. We experimentally validate the effectiveness of these loss functions in Section 4.4.

4 Experiments

All experiments are conducted on a server equipped with 8 NVIDIA A100-80GB GPUs and 512GB of system RAM. We utilize PyTorch (Paszke et al., 2019) as the implementation framework, SGLang (Zheng et al., 2024b) for sampling and DeepSpeed (Aminabadi et al., 2022) for distributed training.

4.1 Settings

Base Models. For our experiments, we employ Qwen2.5-Math-7B-Instruct[2] (Yang et al., 2024) as the base model for PRM training and dataset construction. For code translation, we utilize Qwen2.5-Coder-14B-Instruct[3] (Hui et al., 2024), which exhibits strong performance in converting natural language into executable code.

Dataset Construction. We construct our PRM training dataset through a comprehensive process using SCOPE. Starting with mathematical problems extracted from the openbmb/UltraInteract_sft [4] dataset, we generate 64 solution samples for each problem using Qwen2.5-Math-7B-Instruct. To ensure dataset quality, we carefully filter out problems where solutions unanimously yield either correct or incorrect results, as these extreme cases might introduce training bias. The processing pipeline continues with Qwen2.5-Coder-14B-Instruct translating each reasoning

[2] https://huggingface.co/Qwen/Qwen2.5-Math-7B-Instruct
[3] https://huggingface.co/Qwen/Qwen2.5-Coder-14B-Instruct
[4] https://huggingface.co/datasets/openbmb/UltraInteract_sft

Setting	GSM8K	MATH	Minerva Math	GaoKao 2023 En	Olympiad Bench	College Math	Avg.
Greedy	95.1	80.1	38.2	65.2	39.4	40.6	59.8
Pass@8 (Upper Bound)	97.8	91.5	55.5	79.2	58.1	48.5	71.8
Majority@8	96.7	85.6	43.8	69.6	42.1	41.8	63.3
Math-Shepherd-PRM-7B	95.6	81.3	37.1	64.9	38.2	39.9	59.5
RLHFlow-PRM-Mistral-8B	96.1	83.5	39.3	67.0	41.2	40.8	61.3
RLHFlow-PRM-Deepseek-8B	96.4	83.5	40.8	67.0	39.7	41.1	61.4
Skywork-PRM-1.5B	96.0	83.8	39.0	66.5	39.8	41.2	61.1
Skywork-PRM-7B	96.0	**84.1**	40.3	66.3	40.4	41.8	61.5
EurusPRM-Stage1	95.3	83.1	37.7	65.8	38.5	39.1	59.9
EurusPRM-Stage2	95.1	83.4	37.9	66.1	40.2	39.2	60.3
SCOPE	**96.4**	83.8	**41.2**	**67.1**	**41.2**	**41.9**	**61.9**

Table 1: Performance comparison on the Best-of-8 strategy of the policy model Qwen2.5-Math- 7B-Instruct.

step into executable code, followed by AST-based normalization as detailed in Section 3. For each mathematical problem, we then construct a prefix tree from the 64 normalized solutions and compute Q-values for every node within the tree structure. This enables us to derive both soft and hard labels for each reasoning step based on the computed Q-values. Through this systematic approach, we successfully constructed a comprehensive dataset comprising 509K samples with 4M step labels.

Evaluation. We evaluated our approach using two complementary metrics: (1) Consistent with previous work (Lightman et al., 2023; Luo et al., 2024), we employ the **Best-of-N** (BoN) sampling strategy for evaluation, which selects the highest-scored response from N candidates according to PRM. Using Qwen2.5-Math-7B-Instruct, we sample $N = 8$ responses across multiple mathematical benchmarks: GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021), MinervaMath (Lewkowycz et al., 2022), GaoKao2023En (Liao et al., 2024), OlympiadBench (He et al., 2024), and CollegeMath (Tang et al., 2024). Each candidate solution is scored using the product of step-wise scores from PRM. (2) **ProcessBench** (Zheng et al., 2024a), which is specifically designed to assess error identification in mathematical reasoning, contains four sub-benchmarks: GSM8K, MATH, OlympiadBench, and Omni-MATH (Gao et al., 2024). ProcessBench requires models to either identify the first erroneous step in incorrect solutions or verify the correctnesss in valid solutions.

Baselines. We compare against several recent PRMs: Math-Shepherd-PRM-7B (Wang et al., 2024) which estimates process labels through Monte Carlo simulation, RLHFlow-PRM-Mistral-8B and RLHFlow-PRM-DeepSeek-8B (Xiong et al., 2024) which adopt Math-Shepherd's methodology with different optimization objectives, Skywork-PRM-1.5B and Skywork-PRM-7B (o1 Team, 2024) are two recently released Qwen2.5- Math-based PRMs by Skywork, and EurusPRM-Stage1 and EurusPRM-Stage2 (Cui et al., 2025) which learns process rewards implicitly using ORM-based training.

Training Details. For solution generation, we set both the sampling temperature and top-p to 0.8, with a maximum new token limit of 2048 to ensure comprehensive solution generation. In the code translation phase, we employ a temperature of 0 to ensure deterministic outputs, maintaining a maximum new token limit of 2048. For PRM training, we use a batch size of 256, gradient clipping of 1.0, and the AdamW optimizer (Loshchilov, 2017) with a learning rate of 5e-7 and warm-up ratio of 0.05.

4.2 Main Results

Best-of-N. Table 1 presents a comprehensive comparison of our proposed SCOPE with existing PRMs on the Best-of-N strategy. Our method achieves an average score of 61.9%, showing a 2.4% improvement over Math-Shepherd-PRM-7B (59.5%) while requiring only 5% of its computational cost. SCOPE's performance is competitive with strong baselines such as Skywork-PRM-7B (61.5%) and RLHFlow-PRM-Deepseek-

Model	GSM8K			MATH			OlympiadBench			Omni-MATH			Avg. F1
	Error	Correct	F1	Error	Correct	F1	Error	Correct	F1	Error	Correct	F1	
Math-Shepherd-PRM-7B	32.4	91.7	47.9	18.0	82.0	29.5	15.0	71.1	24.8	14.2	73.0	23.8	31.5
RLHFlow-PRM-Mistral-8B	33.8	99.0	50.4	21.7	72.2	33.4	8.2	43.1	13.8	9.6	45.2	15.8	28.4
RLHFlow-PRM-Deepseek-8B	24.2	98.4	38.8	21.4	80.0	33.8	10.1	51.0	16.9	10.9	51.9	16.9	26.6
Skywork-PRM-1.5B	50.2	71.5	59.0	37.9	65.2	48.0	15.4	26.0	19.3	13.6	32.8	19.2	36.4
Skywork-PRM-7B	61.8	82.9	70.8	43.8	62.2	53.6	17.9	31.9	22.9	14.0	41.9	21.0	42.1
EurusPRM-Stage1	46.9	42.0	44.3	33.3	38.2	35.6	23.9	19.8	21.7	21.9	24.5	23.1	31.2
EurusPRM-Stage2	51.2	44.0	47.3	36.4	35.0	35.7	25.7	18.0	21.2	23.1	19.1	20.9	31.3
SCOPE	60.4	86.5	**71.1**	42.1	71.7	**53.8**	31.0	40.7	**35.2**	25.7	50.6	**34.1**	**48.4**

Table 2: Performance comparison on ProcessBench. **Error** means detection rate in incorrect solutions, **Correct** means verification rate in correct solutions, and **F1** scores combining both measures for balanced evaluation.

8B (61.4%) while maintaining lower training costs.

For context, we also report three reference metrics: greedy decoding (59.8%), majority voting (63.3%), and pass@8 (71.8%, upper bound). While there remains a considerable gap to the theoretical upper bound, SCOPE consistently outperforms the greedy baseline. Note that PRM and majority voting capture complementary aspects of solution quality and can be combined to achieve better performance.

ProcessBench. As a complementary evaluation metric, ProcessBench assesses PRMs' ability to either identify the first erroneous step (error) in incorrect solution or verify the correctness of correct solution (correct). Table 2 reveals a common limitation among existing PRMs, with most models struggling to achieve satisfactory performance on this challenging task.

The previous best performance was achieved by Skywork-PRM-7B with an average F1 score of 42.1%. Our SCOPE significantly improves upon this, reaching 48.4% F1 score, representing a 6.3% absolute improvement. The improvement is more pronounced on challenging datasets: on OlympiadBench, SCOPE achieves an F1 score of 35.2% compared to Skywork-PRM-7B's 22.9%, and on Omni-MATH, our method reaches 34.1% versus 21.0%, demonstrating superior capabilities in error identification. This consistent performance advantage across both simpler and more complex benchmarks suggests that SCOPE has developed a more robust and generalizable understanding of mathematical reasoning processes.

4.3 Impact of Best-of-N Sampling

To investigate the impact of sampling size in Best-of-N setting, we conduct experiments with different N values (4, 8, 16,32,64) across various mathematical benchmarks. As shown in Table 3, increasing N generally leads to improved performance, with MATH steadily rising from 81.7% to 85.1% and GaoKao2023En from 66.1% to 68.7%. However, the improvements become marginal when N increases from 32 to 64, as evidenced by the minimal gains across all benchmarks (e.g., from 97.1% to 97.0% on GSM8K), indicating that simply increasing N cannot continuously boost performance.

Datasets	N=4	N=8	N=16	N=32	N=64
GSM8K	96.5	96.4	96.8	**97.1**	97.0
MATH	81.7	83.8	84.2	84.9	**85.1**
MinervaMath	39.0	41.2	42.3	**43.0**	42.9
GaoKao2023En	66.1	67.1	67.7	68.2	**68.7**
OlympiadBench	39.8	41.2	41.5	41.7	**42.1**
CollegeMath	40.1	41.9	42.5	42.8	**42.9**

Table 3: Performance comparison of different N values in Best-of-N setting.

	Soft Label	Hard Label
Best-of-8 (Avg. Acc)	61.5	61.9
ProcessBench (Avg. F1)	46.8	48.4

Table 4: Comparison on soft and hard labels.

4.4 Soft Labels vs. Hard Labels

As outlined in Section 3.3, PRMs can be trained using either hard labels or soft labels. Table 4 presents a comparative analysis of these two training approaches on both the Best-of-8 and ProcessBench benchmarks. The results consistently demonstrate the superiority of hard labels over soft labels across both evaluation settings, with hard labels achieving performance gains of 0.4% and 1.2% on Best-of-8 and ProcessBench respectively. We attribute the limited performance of soft labels to the noise they introduce into the training process. This limitation

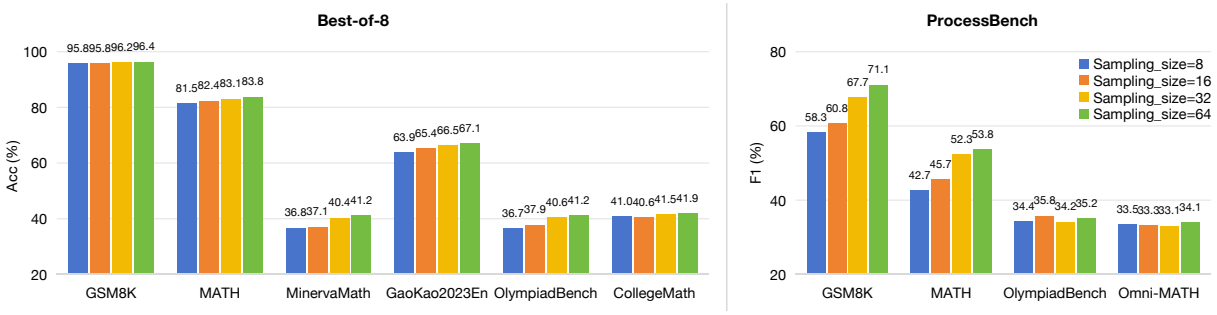

Figure 4: Comparison of different sampling sizes (8, 16, 32, 64) during PRMs dataset construction. We report the F1 scores on ProcessBench.

is particularly evident in complex math problems where correct intermediate steps might receive low soft labels due to the difficulty of reaching the correct final answer, which can introduce confusion during the training process.

4.5 Impact of Sampling Size

To investigate how the number of sampled solutions affects model performance, we conduct experiments with different sampling sizes (8, 16, 32, 64) during PRM dataset construction. Figure 4 presents the results across both Best-of-8 and ProcessBench evaluations. On Best-of-8 (left), we observe modest but consistent improvements across all benchmarks as sampling size increases, with GSM8K accuracy improving from 95.8% to 96.4% and MinervaMatch from 36.8% to 41.2% when scaling from 8 to 64 samples.

The impact on ProcessBench (right) shows varying patterns. GSM8K and MATH demonstrate substantial improvements with increased sampling, achieving 12.8% and 11.1% absolute gains respectively. However, on OlympiadBench and Omni-MATH, the performance gains are limited, likely due to the training data distribution - UltraInteract primarily contains problems from GSM8K and MATH. Based on these observations and considering the computational trade-offs, we chose $N = 64$ as our default sampling size.

4.6 Computational Efficiency

To evaluate the computational efficiency, we sample 100 problems from UltraInteract dataset and conduct PRMs training dataset using different methods. As shown in Figure 5, the GPU hours vary significantly across different approaches. MathShepherd requires 19.8× more GPU hours compared to our method. OmegaPRM and EurusPRM consume 9.8× and 0.6× GPU hours respectively. While EurusPRM shows faster computation, our previous experiments have demonstrated that it yields the poor performance on ProcessBench. In contrast, our approach achieves strong performance while maintaining efficient computation. A detailed breakdown of computational costs for different stages in our method is provided in Appendix A.

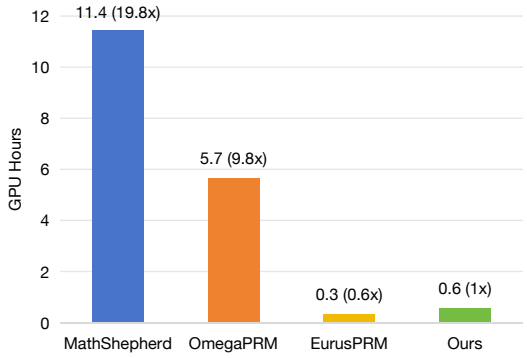

Figure 5: Comparison of time costs (GPU hours) for generating PRM training data across different methods.

5 Conclusion

This paper introduces SCOPE, a novel automatic PRM dataset annotation method that significantly reduces computational costs while maintaining label quality. By translating natural language reasoning steps into code and merging equivalent steps through AST normalization, our approach achieves $O(N)$ complexity compared to previous $O(NMK)$ methods. Using only 5% of the computational resources, we construct a large-scale dataset containing 509K samples, and PRMs trained on our dataset consistently outperform existing approaches on both Best-of-N strategy and ProcessBench, demonstrating SCOPE's effectiveness as a scalable solution for PRM training.

Limitations

While SCOPE demonstrates promising results, several limitations deserve attention: (1) Code Translation Reliability: The quality of our annotations heavily depends on the code LLM's ability to accurately translate natural language reasoning into executable code. Complex mathematical concepts or domain-specific terminology may lead to inaccurate translations, affecting the overall annotation quality. (2) Limited Mathematical Coverage: Our current implementation primarily handles basic arithmetic operations and common mathematical functions. More sophisticated mathematical operations, especially those involving abstract algebra or advanced calculus, may not be adequately captured by our code-based representation.

Future work could focus on developing more robust code translation techniques and expanding the coverage of mathematical operations. Additionally, investigating the integration of domain-specific knowledge and mathematical formalism could further improve the accuracy and applicability of our approach.

References

Alfred Aho, Monica Lam, Ravi Sethi, and Jeffrey D Ullman. 2007. Compilers: Principles, techniques and tools, 2nd editio.

Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, et al. 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.

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

Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu, Qixin Xu, Weize Chen, et al. 2025. Process reinforcement through implicit rewards. *arXiv preprint arXiv:2502.01456*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, et al. 2024. Omni-math: A universal olympiad level mathematic benchmark for large language models. *arXiv preprint arXiv:2410.07985*.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. 2024. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. *arXiv preprint arXiv:2402.14008*.

Dan Hendrycks, Collin Burns, Saurav Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *NeurIPS*.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, et al. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. 2022. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35:3843–3857.

Minpeng Liao, Wei Luo, Chengxi Li, Jing Wu, and Kai Fan. 2024. Mario: Math reasoning with code interpreter output–a reproducible pipeline. *arXiv preprint arXiv:2401.08190*.

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

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

I Loshchilov. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Jianqiao Lu, Zhiyang Dou, WANG Hongru, Zeyu Cao, Jianbo Dai, Yunlong Feng, and Zhijiang Guo. 2024. Autopsv: Automated process-supervised verifier. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu,

Lei Meng, Jiao Sun, et al. 2024. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592*.

Skywork o1 Team. 2024. Skywork-o1 open series. https://huggingface.co/Skywork.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

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

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

Zhengyang Tang, Xingxing Zhang, Benyou Wang, and Furu Wei. 2024. Mathscale: Scaling instruction tuning for mathematical reasoning. *arXiv preprint arXiv:2403.02884*.

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. *arXiv preprint arXiv:2211.14275*.

Eric Wallace, Yizhong Wang, Sujian Li, Sameer Singh, and Matt Gardner. 2019. Do nlp models know numbers? probing numeracy in embeddings. *arXiv preprint arXiv:1909.07940*.

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

Wei Xiong, Hanning Zhang, Nan Jiang, and Tong Zhang. 2024. An implementation of generative prm. https://github.com/RLHFlow/RLHF-Reward-Modeling.

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. 2024. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*.

Fei Yu, Anningzhe Gao, and Benyou Wang. 2023. Outcome-supervised verifiers for planning in mathematical reasoning. *arXiv preprint arXiv:2311.09724*.

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

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

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. 2024b. Sglang: Efficient execution of structured language model programs. *arXiv preprint arXiv:2312.07104*.

A Computational Cost Analysis

Table 5 shows the GPU hours required for each stage of our method. The sampling stage, which involves generating 64 solution samples per problem using Qwen2.5-Math-7B-Instruct, consumes 192 GPU hours. The code translation phase, where Qwen2.5-Coder-14B-Instruct converts natural language steps into executable Python code, requires the most computational resources at 280 GPU hours. Finally, the PRM training stage is relatively efficient, taking 70 GPU hours.

	GPU Hours
Sampling (N = 64)	192
Code Translation	280
PRM Training	70

Table 5: Breakdown of computational costs in different stages of our method.

B Code Translation Prompt

You are a Python expert. I will provide a math problem along with a step-by-step solution. Please present each step of the solution as Python code. Ensure the following requirements are met:

1. Clearly separate each step and save them in different code blocks, using<STEP_START_i> and <STEP_END_i> to separate them, where i represents the i-th step.
2. All calculations should be done in python code. Provide concise reasoning and thinking in the comments of the code.
3. If libraries are required, import them before the first step, using <IMPORT_START> and <IMPORT_END> tags. The most related python packages include 'math', 'sympy', 'scipy', and 'numpy'.
4. Do not use any custom defined functions. Do implement the functionality with the simplest code.
5. Ensure there is corresponding code for each step, even if the code is empty.

Math Problem:
`...(math problem)...`

Solution:
`...(solution)...`

C A Complex Code Translation Example

Question: The quadratic $8x^2 + 12x - 14$ has two real roots. What is the sum of the squares of these roots? Express your answer as a common fraction in lowest terms.

Response	Code

Step 1: We are given the quadratic equation $8x^2 + 12x - 14$ and are asked to find the sum of the squares of its roots.	**<CODE_1>**
	# Given quadratic equation coefficients
	a = 8
	b = 12
	c = -14

Step 2: To find the sum of the squares of the roots, we can use the relationship between the sum of the roots, the product of the roots, and the coefficients of the quadratic equation.	**<CODE_2>** # No calculations needed in this step

Step 3: Let's denote the roots of the quadratic equation as x_1 and x_2. We know that the sum of the roots of a quadratic equation $ax^2 + bx + c = 0$ is given by $x_1 + x_2 = -\frac{b}{a} = -\frac{12}{8}$ and the product of the roots is given by $x_1x_2 = \frac{c}{a} = \frac{-14}{8} = -\frac{7}{4}$.	**<CODE_3>**
	# Calculate the sum and product of the roots
	sum_of_roots = -b / a
	product_of_roots = c / a

Step 4: In order to find the sum of the squares of the roots, which is given by $x_1^2 + x_2^2$, we can use the identity $(x_1 + x_2)^2 = x_1^2 + 2x_1x_2 + x_2^2$ to express the sum of the squares of the roots in terms of the sum and product of the roots.	**<CODE_4>**
	# No calculations needed in this step
	
	<CODE_5>
	# Calculate the square of the sum of the roots
	sum_of_roots_squared = sum_of_roots ** 2

Step 5: Substituting the values we found earlier, we get $(x_1 + x_2)^2 = \left(-\frac{3}{2}\right)^2 = \frac{9}{4}$.	**<CODE_6>**
	# Calculate the sum of the squares of the roots using the identity
	sum_of_squares_of_roots = sum_of_roots_squared - 2 * product_of_roots

Step 6: Then, we can express $x_1^2 + x_2^2$ as $(x_1 + x_2)^2 - 2x_1x_2$, which gives us $x_1^2 + x_2^2 = \frac{9}{4} - 2\left(-\frac{7}{4}\right)$.	**<CODE_7>**
	# Simplify the result
	sum_of_squares_of_roots_simplified = sp.Rational (sum_of_squares_of_roots).limit_denominator()

Step 7: Simplifying this, we get $x_1^2 + x_2^2 = \frac{9}{4} + \frac{14}{4} = \frac{23}{4}$.	
Step 8: Therefore, the sum of the squares of the roots of the quadratic equation $8x^2 + 12x - 14$ is $\frac{23}{4}$.	**<CODE_8>** # No calculations needed in this step

Table 6: An example of natural language reasoning steps and their corresponding code translations for solving a quadratic equation problem.