

# SCOPE: Compress Mathematical Reasoning Steps for Efficient Automated Process Annotation

Huimin Xu<sup>1</sup>, Xin Mao<sup>1</sup>, Feng-Lin Li<sup>2</sup>, Xiaobao Wu<sup>1\*</sup>

Wang Chen<sup>2</sup>, Wei Zhang<sup>3</sup>, Anh Tuan Luu<sup>1\*</sup>

<sup>1</sup>Nanyang Technological University, Singapore

<sup>2</sup>Shopee Pte. Ltd, Singapore, <sup>3</sup>SEA Group, Singapore

{huimin.xu, xin.mao, xiaobao.wu, anhtuan.luu}@ntu.edu.sg

{fenglin.li, chen.wang}@shopee.com, terry.zhang@sea.com

## 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 COverpression 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 196K 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<sup>1</sup>.

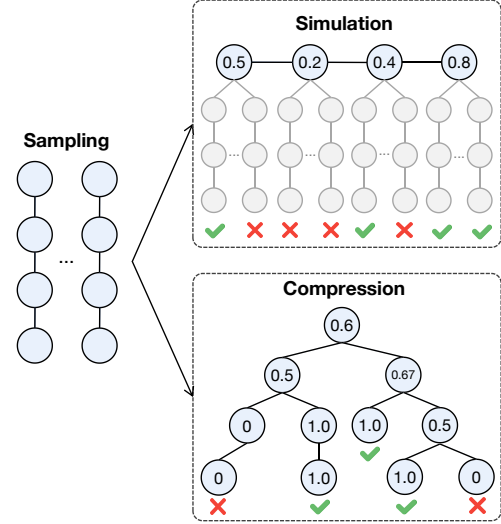


Figure 1: Comparison of PRMs training data construction. Simulation-based methods require numerous completions solely for Q-value estimation, with these completions (gray nodes) being discarded without contributing to training. Our compression-based approach eliminates such data waste by merging equivalent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

## 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; Wu, 2025). 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).

To alleviate this limitation, recent studies explore simulation-based methods. For instance, Mathshepherd (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 Mathshepherd eliminates human annotation requirements, it incurs high computational complexity of  $O(NMK)$ , where  $K$  denotes the average step count of solutions. Recent studies have shown

\*Corresponding Authors.

<sup>1</sup>Our code, data, and models are available at <https://github.com/Anna7355/SCOPE>.

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 Figure 1), resulting in wasted data.

In this paper, we introduce **Step COmpression for Process Estimation (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 samples numerous solutions for each problem, then merges equivalent solution steps to construct a prefix tree (Trie), as shown in Figure 1. For each node in the tree, its Q-value is 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 embeddings 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 196K samples with 1.4M 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 the Best-of-N strategy and the ProcessBench (Zheng et al., 2024a) evaluation.

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 construct a new PRM training dataset containing 196K samples and 1.4M step-level labels, while requiring only 5% of the computational resources used by MathShepherd.
- Extensive experiments show that PRMs trained on our dataset consistently outperform 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.

**Question:** KK climbed a 30-foot ladder 20 times, and Tom climbed a 26-foot ladder 15 times. What is the total length they climbed in inches?

**Solution:**

Step 1: KK climbed a 30-foot ladder 20 times. The total length he climbed is:  $30 \times 20 = 600$  feet.

Step 2: Tom's ladder is 26 feet tall. He climbed it 15 times, so the total length is  $26 \times 15 = 390$  feet.

Step 3: To find the total length both workers climbed, add the lengths climbed by KK and Tom.

Step 4: Therefore, the answer is:  $600 + 390 = 990$  feet.

**Code:**

```
<Code 1>
kk_height = 30
kk_climbs = 20
kk_len = kk_height * kk_climbs

<Code 2>
tom_height = 26
tom_climbs = 15
tom_len = tom_height * tom_climbs

<Code 3>
total_len = kk_len + tom_len

<Code 4>
result = 990
```

**Normalized Code:**

```
<Code 1>
var0 = 30
var1 = 20
var2 = var0 * var1

<Code 2>
var3 = 26
var4 = 15
var5 = var3 * var4

<Code 3>
var6 = var2 + var5

<Code 4>
var7 = 990
```

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 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.

## 3 Method

In this section, we present SCOPE, a novel approach for automatic PRMs 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 perform AST normalization to standardize code syntax for accurate equivalence matching. (3) Next, we apply step compression by merging normalized steps into a prefix tree. (4) Finally, we describe our PRMs training details.

### 3.1 Code Translation

As discussed in Section 1, the key challenge of SCOPE lies in efficiently identifying and merging equivalent reasoning steps. A naive approach based on exact string matching fails to recognize equivalent steps expressed differently (e.g., “multiply 5 by 3” vs “calculate  $5 \times 3$ ”), leading to an overly sparse compression space. While edit distance or sentence embeddings 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 in 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 arith-





### 3.4 PRMs 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 (Wang et al., 2024), we explore two strategies to estimate the label  $y_{s_i}$  for each 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 one solution path through this step reaches the correct answer:

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

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:

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

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:

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

where  $\hat{y}_{s_i}$  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:

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

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.

## 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<sup>2</sup> (Yang et al., 2024) as the base model for PRMs training and dataset construction. For code translation, we utilize Qwen2.5-Coder-32B-Instruct<sup>3</sup> (Hui et al., 2024), which exhibits strong performance in converting natural language into executable code.

**Dataset Construction.** We construct our PRMs training dataset using the SCOPE pipeline. Starting from math problems in the UltraInteract dataset<sup>4</sup>, we generate 64 solutions per problem using Qwen2.5-Math-7B-Instruct. Following prior work, we remove problems whose model confidence<sup>5</sup> equals 0 or 1, as such extremes may introduce training bias. We also observe that the distribution of model confidence has a strong impact on PRMs performance; thus, we retain only problems with model confidence greater than 0.75 to ensure label reliability.

For each retained problem, Qwen2.5-Coder-32B-Instruct translates reasoning steps into code, followed by AST-based normalization. We then construct a prefix tree from the 64 normalized solutions and compute Q-values for each node. The final dataset contains 196K samples and 1.4M 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 the 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 the PRM. (2) **ProcessBench** (Zheng et al., 2024a), which is specifically designed to

<sup>2</sup><https://huggingface.co/Qwen/Qwen2.5-Math-7B-Instruct>

<sup>3</sup><https://huggingface.co/Qwen/Qwen2.5-Coder-32B-Instruct>

<sup>4</sup>[https://huggingface.co/datasets/openbmb/UltraInteract\\_sft](https://huggingface.co/datasets/openbmb/UltraInteract_sft)

<sup>5</sup>Defined as the proportion of correct solutions among the 64 generated by the base model.

Setting	GSM8K	MATH	Minerva Math	GaoKao 2023 En	Olympiad Bench	College Math	Avg.
Greedy	95.5	83.0	34.6	64.2	38.2	46.3	60.3
Pass@8 (Upper Bound)	97.8	91.8	46.7	79.7	59.4	52.5	71.3
Majority@8	96.5	86.9	40.1	70.4	46.2	47.8	64.7
Math-Shepherd-PRM-7B	96.2	81.7	33.8	63.6	39.1	45.5	60.0
RLHFlow-PRM-Mistral-8B	96.0	85.7	37.5	70.9	43.3	47.6	63.5
RLHFlow-PRM-Deepseek-8B	96.7	85.6	38.2	69.9	44.3	47.4	63.7
EurusPRM-Stage1	95.1	83.4	37.9	66.1	40.2	39.2	60.0
EurusPRM-Stage2	95.8	83.1	37.7	65.8	38.5	41.0	60.3
Skywork-PRM-1.5B	96.2	86.3	38.2	70.6	43.0	48.1	63.7
Skywork-PRM-7B	96.4	86.1	<b>39.1</b>	70.9	42.8	47.9	63.9
*Qwen2.5-Math-7B-PRM800K	96.5	86.5	38.1	70.5	43.5	48.2	63.0
<b>SCOPE</b>	<b>96.7</b>	<b>87.7</b>	38.2	71.9	<b>46.8</b>	<b>48.3</b>	<b>64.9</b>
- w/o AST normalization	96.5	87.3	38.2	<b>72.2</b>	45.6	48.3	64.7
- w/o code translation	96.6	87.0	36.4	70.6	45.5	48.0	64.0
- Step Replacement	96.6	87.3	37.1	71.0	45.3	48.3	64.3
- Step Skipping	96.7	87.7	37.1	71.7	45.8	48.3	64.6

Table 1: Performance comparison on the Best-of-8 strategy. \* is trained on high-quality manually annotated process-level data (PRM800K); all others are trained using automatically constructed datasets.

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 correctness in valid solutions.

**Baselines.** We compare against 7B-scale PRMs: Math-Shepherd-PRM-7B (Wang et al., 2024) estimates process labels through Monte Carlo simulation. RLHFlow-PRM-Mistral-8B and RLHFlow-PRM-DeepSeek-8B (Xiong et al., 2024) adopt Math-Shepherd’s methodology with different optimization objectives. EurusPRM-Stage1 and EurusPRM-Stage2 (Cui et al., 2025) learn process rewards implicitly from ORM-based training. Skywork-PRM-1.5B and Skywork-PRM-7B (ol Team, 2024) are two recently released Qwen2.5-Math-based PRMs by Skywork. Finally, we include Qwen2.5-Math-7B-PRM800K (Zheng et al., 2024a) as a strong baseline. It is fine-tuned on the PRM800K dataset (Lightman et al., 2023), a high-quality, manually annotated corpus of 265K process-level samples.

For fairness, we do not compare against Qwen2.5-Math-PRM (Zheng et al., 2024a), which uses a 72B critic model to supervise 1.5M process-level annotations. Their focus is on

distilling critic capabilities from large models, while our work aims to reduce annotation cost.

**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-8 strategy. SCOPE achieves an average accuracy of 64.9%, outperforming Math-Shepherd-PRM-7B by 4.9% while requiring only 5% of its computational cost. Notably, SCOPE also surpasses Qwen2.5-Math-7B-PRM800K (63.0%), which is trained on high-quality, manually annotated process-level data, demonstrating the effectiveness of our approach.

For context, we also report three reference metrics: greedy decoding (60.3%), majority voting (64.7%), and pass@8 (71.3%, upper bound). SCOPE is the only method that outperforms

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
*Qwen2.5-Math-7B-PRM800K	53.1	95.3	68.2	48.0	90.1	<b>62.6</b>	35.7	87.3	<b>50.7</b>	29.8	86.1	<b>44.3</b>	<b>56.5</b>
<b>SCOPE</b>	59.9	86.0	<b>70.6</b>	50.8	74.9	60.6	35.7	59.3	44.6	31.4	62.2	41.7	54.4
- w/o AST Normalization	61.4	85.5	71.4	51.9	74.6	61.2	34.3	56.6	42.8	29.8	57.3	39.2	53.6
- w/o Code Translation	56.5	83.9	67.6	47.0	75.6	57.9	32.7	54.9	41.0	26.5	56.8	36.1	50.6
- Step Replacement	53.6	87.6	66.5	35.5	83.0	49.8	23.8	69.3	35.4	16.5	68.5	26.6	44.6
- Step Skipping	56.5	90.2	69.5	38.0	84.5	52.5	24.1	74.9	36.4	15.9	71.0	26.0	44.6

Table 2: Performance comparison on ProcessBench. \* is trained on high-quality manually annotated process-level data (PRM800K); all others are trained using automatically constructed datasets.

Majority@8, a very strong baseline that reflects the collective correctness of multiple model outputs.

**ProcessBench.** As a complementary evaluation metric, ProcessBench assesses PRMs’ ability to either identify the first erroneous step in incorrect solution or verify the correctness of correct solution. As shown in Table 2, most existing PRMs trained on automatically annotated data struggle on this benchmark, with average F1 scores typically below 45%. The strongest performance is achieved by Qwen2.5-Math-7B-PRM800K (56.5% F1), which is trained on a high-quality manually labeled process-level dataset (PRM800K). Our method, SCOPE, achieves 54.4%, closely approaching this supervised upper bound, while being fully automatic.

### 4.3 Ablation on Key Components

To evaluate the impact of core components in SCOPE, we conduct two ablation studies:

- **w/o Code Translation:** This variant skips the code translation step and merges reasoning steps directly based on natural language.
- **w/o AST Normalization:** This variant performs code translation but skips AST-based normalization, merging code blocks in their raw form.

We begin by analyzing compression efficiency. We define the compression rate as the ratio of compressed nodes in the Trie to the total number of reasoning steps before compression. A higher compression rate indicates less effective compression,

i.e., more redundancy remains. As shown in Figure 4, w/o code translation leads to a high compression rate of 92.0%, suggesting that natural language steps are too diverse to merge effectively. Introducing code translation reduces the rate to 79.5%, while further applying AST normalization brings it down to 65.5%.

We then examine the impact on downstream performance. From Table 1 and Table 2, we observe: (1) removing code translation leads to a 0.9% drop in Best-of-8 accuracy and a 3.8% drop in ProcessBench F1. (2) Removing AST normalization also causes slight performance drops, indicating its positive contribution. These results confirm that both code translation and AST normalization are essential for improving compression quality and PRM performance.

### 4.4 Comparison of Step Compression Strategies

Beyond evaluating the core components of SCOPE, we further investigate whether alternative strategies can improve the compression process, especially for comment-only steps. These steps, which account for 27.3% of all reasoning steps, are treated as distinct nodes in our default setting. But is this the best design choice? We design two additional strategies:

- **Step Replacement:** Replace all comment-only steps with a generic placeholder string like  $step_i = \text{"onlycomment"}$ . This allows more aggressive merging.
- **Step Skipping:** Omit comment-only steps from the Trie entirely. During Q-value com-

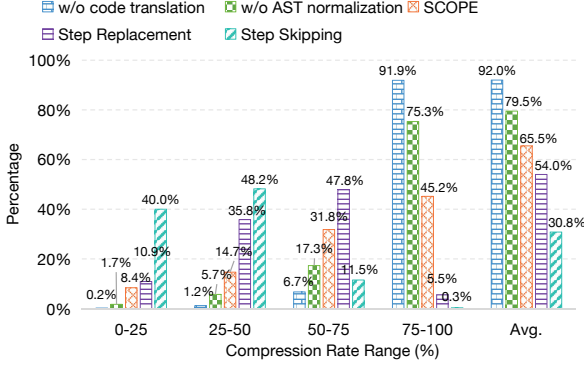


Figure 4: Distribution of compression rates across different SCOPE variants. Compression Rate = compressed nodes in Trie / raw step count.

	Soft Label	Hard Label
Best-of-8 (Avg. Acc)	64.6	64.9
ProcessBench (Avg. F1)	52.8	54.4

Table 3: Comparison on soft and hard labels.

putation, these steps inherit the score of their preceding step to preserve label continuity.

Both strategies decrease the compression ratio. As shown in Figure 4, Step Replacement and Step Skipping lead to compression rates of 54.0% and 30.8%, respectively—compared to 65.5% in the default setting. However, this decrease in compression ratio comes at a cost. As shown in Table 1 and Table 2, both strategies result in lower performance on Best-of-8 and ProcessBench, despite achieving higher compression. These results suggest that over-compression harms PRM performance. Treating comment-only steps as distinct nodes, although computationally less efficient, provides better alignment supervision by preserving the reasoning structure.

#### 4.5 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 (Wang et al., 2024) requires 19.8× more GPU hours compared to our method. OmegaPRM (Luo et al., 2024) and EurushPRM (Cui et al., 2025) consume 9.8× and 0.6× GPU hours respectively. While EurushPRM shows faster computation, our previous experiments have demonstrated that it yields the poor performance on Best-of-8 and ProcessBench. In contrast, our

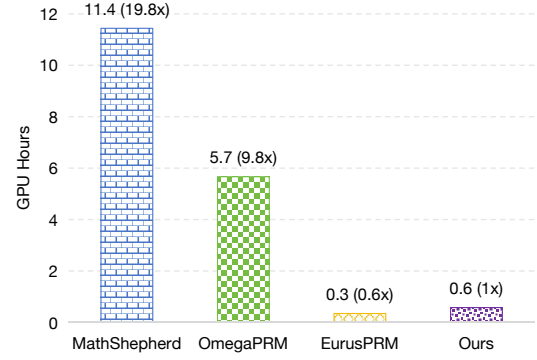


Figure 5: Comparison of time costs (GPU hours) for generating PRM training data across different methods.

approach achieves strong performance while maintaining efficient computation. A detailed breakdown of computational costs for different stages in our method is provided in Appendix B.

#### 4.6 Soft Labels vs. Hard Labels

As outlined in Section 3.4, PRMs can be trained using either hard labels or soft labels. Table 3 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 0.3% and 1.6% 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 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.

### 5 Conclusion

This paper introduces SCOPE, a novel automatic PRMs 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 196K 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.



## Acknowledgement

This research/project is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG2-TC-2022-005). We also wish to extend their heartfelt gratitude to the Sea AI Lab for their generous support in providing the necessary equipment and computational resources critical for the successful completion of this research.

## 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 (Pan et al., 2024; Wu et al., 2020, 2024a,b,c) 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. [Olympiad-bench: 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, Jiayi 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>.
- Fengjun Pan, Xiaobao Wu, Zongrui Li, and Anh Tuan Luu. 2024. [Are LLMs good zero-shot fallacy classifiers?](#) In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 14338–14364, Miami, Florida, USA. Association for Computational Linguistics.
- 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.
- Xiaobao Wu. 2025. [Sailing ai by the stars: A survey of learning from rewards in post-training and test-time scaling of large language models](#). *arXiv preprint arXiv:2505.02686*.
- Xiaobao Wu, Chunping Li, Yan Zhu, and Yishu Miao. 2020. [Short text topic modeling with topic distribution quantization and negative sampling decoder](#). In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1772–1782, Online.
- Xiaobao Wu, Thong Nguyen, and Anh Tuan Luu. 2024a. [A survey on neural topic models: Methods, applications, and challenges](#). *Artificial Intelligence Review*.
- Xiaobao Wu, Liangming Pan, William Yang Wang, and Anh Tuan Luu. 2024b. [AKEW: Assessing knowledge editing in the wild](#). In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 15118–15133, Miami, Florida, USA. Association for Computational Linguistics.
- Xiaobao Wu, Liangming Pan, Yuxi Xie, Ruiwen Zhou, Shuai Zhao, Yubo Ma, Mingzhe Du, Rui Mao, Anh Tuan Luu, and William Yang Wang. 2024c. [AntiLeak-Bench: Preventing data contamination by automatically constructing benchmarks with updated real-world knowledge](#). *arXiv preprint arXiv:2412.13670*.
- 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 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)...

## B Complexity Analysis and Time Cost Evaluation

In this appendix, we provide a more rigorous analysis of the theoretical time complexity of SCOPE, alongside a breakdown of the actual GPU and CPU time costs for each stage in our pipeline.

### B.1 Stage-wise Complexity Analysis

SCOPE consists of the following stages: (1) Sampling of solutions, (2) Code translation, (3) AST normalization, and (4) Step compression. We analyze the complexity of each component below:

- **Solution Sampling.** For each math problem, we sample  $N$  complete solutions using a mathematical LLM. Since each solution is generated in a single forward pass, the overall sampling complexity is  $O(N)$ .
- **Code Translation.** Each solution is translated into a sequence of code steps using a code LLM in a single call. Although this is the most time-consuming stage in practice due to the use of a large LLM, the complexity remains  $O(N)$ , as each solution corresponds to one model invocation.

- **AST Normalization.** After code translation, each of the  $K$  steps in all  $N$  solutions is normalized using AST-based transformations. As each normalization call processes a complete solution at once, this stage also has linear complexity  $O(N)$ . Moreover, it runs entirely on CPU and typically completes in under 10 seconds, making its cost negligible in practice.
- **Step Compression.** Normalized code sequences are merged into a prefix tree (Trie). Trie construction requires a single pass through all steps, resulting time complexity of  $O(N)$ . In practice, this stage also runs on CPU and completes within 5 seconds.

All stages in SCOPE have linear time complexity with respect to the number of sampled solutions. Therefore, the end-to-end complexity of SCOPE remains  $O(N)$ .

### B.2 Actual Time Cost Breakdown

Table 4 reports time complexity and empirical GPU hours required for each stage of our method.

## C AST Structure

Figure 6 illustrates the Abstract Syntax Tree (AST) representation used in our code normalization process. The AST transforms code into a hierarchical structure that prioritizes semantic relationships over syntactic details. In this representation, the root Module node connects to various Assign operations that define variables. Each variable assignment includes nodes for the target identifier (Name with id attribute) and its value, which may be a Constant or a binary operation (BinOp). Binary operations are represented with explicit operator nodes (Add, Mult) connecting their operands. This AST-based approach enables our system to recognize that statements like “`kk_len = kk_height * kk_climbs`” and “`len = height * climbs`” are structurally equivalent after normalization, forming the foundation for our step compression algorithm.

## D A Complex Code Translation Example

Table 5 shows a code translation example for a complex problem, where three steps are comments only.

Stage	Time Complexity	GPU Hours	Comment
Solution Sampling	$O(N)$	192	Using Qwen2.5-Math-7B
Code Translation	$O(N)$	280	Using Qwen2.5-Coder-32B
AST Normalization	$O(N)$	~0	< 10 seconds on CPU
Step Compression	$O(N)$	~0	< 5 seconds on CPU
Total	$O(N)$	~542	Efficient end-to-end pipeline

Table 4: Breakdown of time complexity and costs in different stages of our method.

**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
<p><b>Step 1:</b> We are given the quadratic equation <math>8x^2 + 12x - 14</math> and are asked to find the sum of the squares of its roots.</p> <p><b>Step 2:</b> 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.</p> <p><b>Step 3:</b> Let's denote the roots of the quadratic equation as <math>x_1</math> and <math>x_2</math>. We know that the sum of the roots of a quadratic equation <math>ax^2 + bx + c = 0</math> is given by <math>x_1 + x_2 = -\frac{b}{a} = -\frac{12}{8}</math> and the product of the roots is given by <math>x_1x_2 = \frac{c}{a} = \frac{-14}{8} = -\frac{7}{4}</math>.</p> <p><b>Step 4:</b> In order to find the sum of the squares of the roots, which is given by <math>x_1^2 + x_2^2</math>, we can use the identity <math>(x_1 + x_2)^2 = x_1^2 + 2x_1x_2 + x_2^2</math> to express the sum of the squares of the roots in terms of the sum and product of the roots.</p> <p><b>Step 5:</b> Substituting the values we found earlier, we get <math>(x_1 + x_2)^2 = \left(-\frac{3}{2}\right)^2 = \frac{9}{4}</math>.</p> <p><b>Step 6:</b> Then, we can express <math>x_1^2 + x_2^2</math> as <math>(x_1 + x_2)^2 - 2x_1x_2</math>, which gives us <math>x_1^2 + x_2^2 = \frac{9}{4} - 2\left(-\frac{7}{4}\right)</math>.</p> <p><b>Step 7:</b> Simplifying this, we get <math>x_1^2 + x_2^2 = \frac{9}{4} + \frac{14}{4} = \frac{23}{4}</math>.</p> <p><b>Step 8:</b> Therefore, the sum of the squares of the roots of the quadratic equation <math>8x^2 + 12x - 14</math> is <math>\frac{23}{4}</math>.</p>	<p><b>&lt;CODE_1&gt;</b>  # Given quadratic equation coefficients  a = 8  b = 12  c = -14</p> <p><b>&lt;CODE_2&gt;</b>  # No calculations needed in this step, just setting up for next steps</p> <p><b>&lt;CODE_3&gt;</b>  # Calculate the sum and product of the roots  sum_of_roots = -b / a  product_of_roots = c / a</p> <p><b>&lt;CODE_4&gt;</b>  # No calculations needed in this step, just setting up for next steps</p> <p><b>&lt;CODE_5&gt;</b>  # Calculate the square of the sum of the roots  sum_of_roots_squared = sum_of_roots ** 2</p> <p><b>&lt;CODE_6&gt;</b>  # 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</p> <p><b>&lt;CODE_7&gt;</b>  # Simplify the result  sum_of_squares_of_roots_simplified = sp.Rational(sum_of_squares_of_roots).limit_denominator()</p> <p><b>&lt;CODE_8&gt;</b>  # No calculations needed in this step, just stating the final answer</p>

Table 5: An example of natural language reasoning steps and their corresponding code translations for solving a quadratic equation problem.



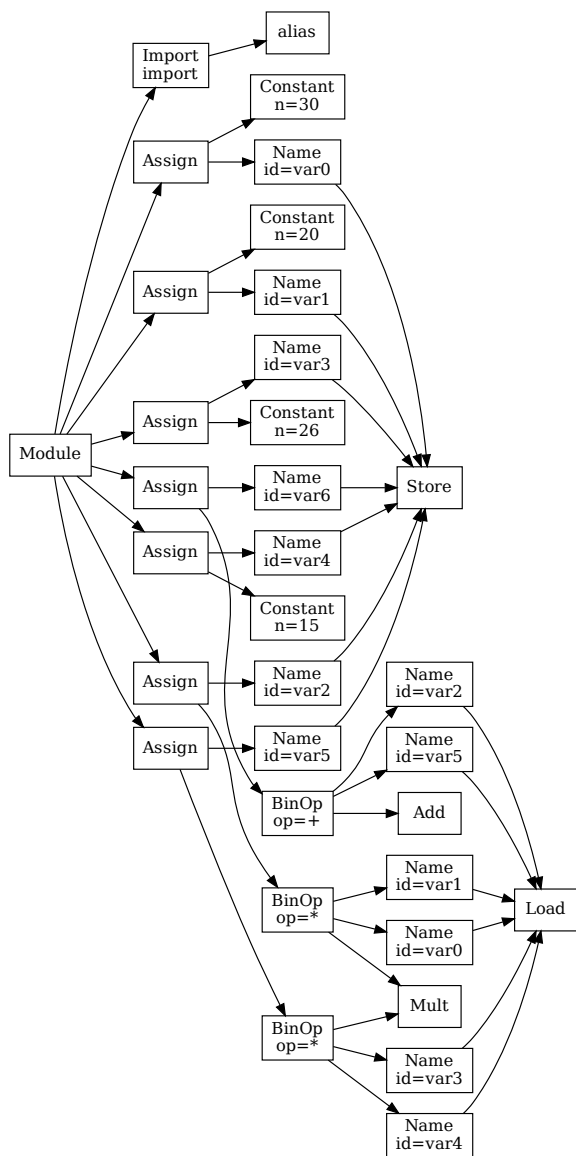


Figure 6: AST structure of the ladder problem code.