Math-Shepherd: Verify and Reinforce LLMs Step-by-step without Human Annotations

Peiyi Wang[1] **Lei Li**[3] **Zhihong Shao**[4] **Runxin Xu**[2] **Damai Dai**[1] **Yifei Li**[5]
Deli Chen[2] **Yu Wu**[2] **Zhifang Sui**[1]
[1]State Key Laboratory of Multimedia Information Processing, School of Computer Science, Peking University.
[2]DeepSeek-AI [3]The University of Hong Kong
[4]Tsinghua University [5]The Ohio State University
{wangpeiyi9979, nlp.lilei}@gmail.com
li.14042@osu.edu szf@pku.edu.cn

Abstract

In this paper, we present an innovative process-oriented math process reward model called **MATH-SHEPHERD**, which assigns a reward score to each step of math problem solutions. The training of MATH-SHEPHERD is achieved using automatically constructed process-wise supervision data, breaking the bottleneck of heavy reliance on manual annotation in existing work. We explore the effectiveness of MATH-SHEPHERD in two scenarios: 1) *Verification*: MATH-SHEPHERD is utilized for reranking multiple outputs generated by Large Language Models (LLMs); 2) *Reinforcement Learning (RL)*: MATH-SHEPHERD is employed to reinforce LLMs. With MATH-SHEPHERD, a series of open-source LLMs demonstrates exceptional performance. For instance, process RL with MATH-SHEPHERD significantly enhances Mistral-7B (77.9%→84.1% on GSM8K and 28.6%→33.0% on MATH). The accuracy can be further improved to 89.1% and 43.5% on two benchmarks with verification of MATH-SHEPHERD. We believe that automatic process supervision holds significant potential for the future evolution of LLMs.

1 Introduction

Large language models (LLMs) have demonstrated remarkable capabilities across various tasks (Park et al., 2023; Kaddour et al., 2023; Song et al.; Li et al., 2023a; Wang et al., 2023a; Chen et al., 2023; Zheng et al., 2023; Wang et al., 2023c), However, even the most advanced LLMs face challenges in complex multi-step mathematical reasoning problems (Lightman et al., 2023; Huang et al., 2023). To address this issue, prior research has explored different methodologies, such as pre-training (Azerbayev et al., 2023), fine-tuning (Luo et al., 2023; Yu et al., 2023b; Wang et al., 2023b), prompting (Wei et al., 2022; Fu et al., 2022), and verification (Wang et al., 2023d; Li et al., 2023b; Zhu et al., 2023; Leviathan et al., 2023). Among these techniques, verification has recently emerged as a favored method. The motivation behind verification is that relying solely on the top-1 result may not always produce reliable outcomes. A verification model can rerank candidate responses, ensuring higher accuracy and consistency in the outputs of LLMs. In addition, a good verification model can also offer invaluable feedback for further improvement of LLMs (Uesato et al., 2022; Wang et al., 2023b; Pan et al., 2023).

The verification models generally fall into the outcome reward model (ORM) (Cobbe et al., 2021; Yu et al., 2023a) and process reward model (PRM) (Li et al., 2023b; Khalifa et al., 2023; Uesato et al., 2022; Lightman et al., 2023; Ma et al., 2023). The ORM assigns a confidence score based on the entire generation sequence, whereas the PRM evaluates the reasoning path step-by-step. PRM is advantageous due to several compelling reasons. One major benefit is its ability to offer precise feedback by identifying the specific location of any errors that may arise, which is a valuable signal in reinforcement learning and automatic correction. Besides, The PRM exhibits similarities to human behavior when assessing a reasoning problem. If any steps contain an error, the final result is more likely to be incorrect, mirroring the way human judgment works. However, gathering data to train a PRM can be an arduous process. (Uesato et al., 2022) and (Lightman et al., 2023) utilize human annotators to provide process supervision annotations, enhancing the performance of PRM. Nevertheless, annotation by humans, particularly for intricate multi-step reasoning tasks that require advanced annotator skills, can be quite costly, which hinders the advancement and practical application of PRM.

To tackle the problem, in this paper, we propose an automatic process annotation framework. Inspired by Monte Carlo Tree Search (Kocsis & Szepesvári, 2006; Coulom, 2006; Silver et al., 2016; Świechowski et al., 2023), we define the

Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 9426–9439

quality of an intermediate step as its potential to deduce the correct final answer. By leveraging the correctness of the answer, we can automatically gather step-wise supervision. Specifically, given a math problem with a golden answer and a step-by-step solution, to achieve the label of a specific step, we utilize a fine-tuned LLM to decode multiple subsequent reasoning paths from this step. We further validate whether the decoded final answer matches with the golden answer. If a reasoning step can deduce more correct answers than another, it would be assigned a higher correctness score.

We use this automatic way to construct the training data for MATH-SHEPHERD, and verify our ideas on two widely used mathematical benchmarks, GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021). We explore the effectiveness of MATH-SHEPHERD in two scenarios: 1) verification: MATH-SHEPHERD is utilized for reranking multiple outputs generated by LLMs; 2) reinforcement learning: MATH-SHEPHERD is employed to reinforce LLMs with step-by-step PPO. With the verification of MATH-SHEPHERD, a series of open-source LLMs from 7B to 70B demonstrates exceptional performance. For instance, the step-by-step PPO with MATH-SHEPHERD significantly improves the accuracy of Mistral-7B (77.9%→84.1% on GSM8K and 28.6%→33.0% on MATH). The accuracy can be further enhanced to 89.1% and 43.5% on GSM8K and MATH with verification. DeepSeek 67B (DeepSeek, 2023) achieves accuracy rates of 93.3% on the GSM8K dataset and 48.1% on the MATH dataset with verification of MATH-SHEPHERD. To the best of our knowledge, these results are unprecedented for open-source models that do not rely on additional tools.

Our main contributions are as follows: 1) We propose a framework to automatically construct process supervision datasets without human annotations for math reasoning tasks; 2) We evaluate our method on both step-by-step verification and reinforcement learning scenarios. Extensive experiments on two widely used mathematical benchmarks - GSM8K and MATH, in addition to a series of LLMs ranging from 7B to 70B, demonstrate the effectiveness of our method; 3) We empirically analyze the key factors for training high-performing process reward models, shedding light on future directions toward improving reasoning capability with automatic step-by-step verification and supervision.

2 Related Works

Improving and eliciting mathematical reasoning abilities of LLMs. Mathematical reasoning tasks are one of the most challenging tasks for LLMs. Researchers have proposed various methods to improve or elicit the mathematical reasoning ability of LLMs, which can be broadly divided into three groups: 1) *pre-training*: The pre-training methods (OpenAI, 2023; Anil et al., 2023; Touvron et al., 2023; Azerbayev et al., 2023) pre-train LLMs on a vast of datasets that are related to math problems, such as the Proof-Pile and ArXiv (Azerbayev et al., 2023) with a simple next token prediction objective. 2) *fine-tuning*: The fine-tuning methods (Yu et al., 2023b; Luo et al., 2023; Yue et al., 2023; Wang et al., 2023b; Gou et al., 2023) can also enhance the mathematical reasoning ability of LLMs. The core of fine-tuning usually lies in constructing high-quality question-response pair datasets with a chain-of-thought reasoning process. and 3) *prompting*: The prompting methods (Wei et al., 2022; Zhang et al., 2023; Fu et al., 2022; Bi et al., 2023) aim to elicit the mathematical reasoning ability of LLMs by designing prompting strategy without updating the model parameters, which is very convenient and practical.

Mathematical reasoning verification for LLMs. Except for directly improving and eliciting the mathematical reasoning potential of LLMs, the reasoning results can be boosted via an extra verifier for selecting the best answer from multiple decoded candidates. There are two primary types of verifiers: the Outcome Reward Model (ORM) and the Process Reward Model (PRM). The ORM allocates a score to the entire solution while the PRM assigns a score to each individual step in the reasoning process. Recent findings by (Lightman et al., 2023) suggest that PRM outperforms ORM. In addition to verification, reward models can offer invaluable feedback for further training of generators (Uesato et al., 2022; Pan et al., 2023). Compared to ORM, PRM provides more detailed feedback, demonstrating greater potential to enhance generator (Wu et al., 2023). However, training a PRM requires access to expensive human-annotated datasets (Uesato et al., 2022; Lightman et al., 2023), which hinders the advancement and practical application of PRM. Therefore, we aim to build a PRM for mathematical reasoning without human annotation, and we explore the effective-

ness of the automatic PRM with both verification and reinforcement learning scenarios.

3 Methodology

In this section, we first present our task formulation to evaluate the performance of reward models (§3.1). Subsequently, we outline two typical categories of reward models, ORM and PRM(§3.2). Then, we introduce our methodology to automatically build the training dataset for PRM(§3.3), breaking the bottleneck of heavy reliance on manual annotation in existing work (Uesato et al., 2022; Lightman et al., 2023).

3.1 Task Formulation

We evaluate the performance of the reward model in two scenarios:

Verification Following (Lightman et al., 2023), we consider a best-of-N selection evaluation paradigm. Specifically, given a problem p in the testing set, we sample N candidate solutions from a generator. These candidates are then scored using a reward model, and the highest-scoring solution is selected as the final answer. An enhanced reward model elevates the likelihood of selecting the solution containing the correct answer, consequently raising the success rate in solving mathematical problems for LLMs.

Reinforcement learning We also use the automatically constructed PRM to supervise LLMs with step-by-step RL. In this scenario, we evaluate the accuracy of the LLMs' greedy decoding output. An enhanced reward model is instrumental in training higher-performing LLMs.

3.2 Reward Models

ORM Given a mathematical problem p and its solution s, ORM ($P \times S \rightarrow \mathbb{R}$) assigns a single real-value to s to indicate whether s is correct. ORM is usually trained with a cross-entropy loss (Cobbe et al., 2021; Li et al., 2023b):

$$\mathcal{L}_{ORM} = -(y_s \log r_s + (1 - y_s) \log(1 - r_s)), \quad (1)$$

where y_s is the golden answer of the solution s, $y_s = 1$ if s is correct, otherwise $y_s = 0$. r_s is the sigmoid score of s assigned by ORM. The success of the reward model hinges on the effective construction of the high-quality training dataset. As the math problem usually has a certain answer, we can automatically construct the training set of ORM by two steps: 1) sampling some candidate solutions for a problem from a generator; 2) assigning the label to each sampling solution by checking whether its answer is correct. Although *false positives* solutions that reach the correct answer with incorrect reasoning will be misgraded, previous studies have proven that it is still effective for training a good ORM (Lightman et al., 2023; Yu et al., 2023a).

PRM Take a step further, PRM ($P \times S \rightarrow \mathbb{R}^+$) assigns a score to each reasoning step of s, which is usually trained with:

$$\mathcal{L}_{PRM} = -\sum_{i=1}^{K} y_{s_i} \log r_{s_i} + (1 - y_{s_i}) \log(1 - r_{s_i}), \quad (2)$$

where y_{s_i} is the golden answer of s_i (the i-th step of s), r_{s_i} is the sigmoid score of s_i assigned by PRM and K is the number of reasoning steps for s. (Lightman et al., 2023) also conceptualizes the PRM training as a three-class classification problem, in which each step is classified as either 'good', 'neutral', or 'bad'. In this paper, we found that there is not much difference between the binary and the three classifications, and we regard PRM training as the binary classification. Compared to ORM, PRM can provide more detailed and reliable feedback (Lightman et al., 2023). However, there are currently no automated methods available for constructing high-quality PRM training datasets. Previous works (Uesato et al., 2022; Lightman et al., 2023) typically resort to costly human annotations. The annotation cost invariably impedes both the development and application of PRM.

3.3 Automatic Process Annotation

In this section, we propose an automatic process annotation framework to mitigate the annotation cost issues associated with PRM. We first define the quality of a reasoning step, followed by the introduction of our solution that obviates the necessity for human annotation.

3.3.1 Definition

Inspired by Monto Carlo Tree Search (Kocsis & Szepesvári, 2006; Coulom, 2006; Silver et al., 2016; Świechowski et al., 2023), we define the quality of a reasoning step as *its potential to deduce the correct answer.* This criterion stems from the primary objective of the reasoning process, which essentially is a cognitive procedure aiding humans or intelligent agents in reaching a well-founded outcome (Huang & Chang, 2023). Therefore, a

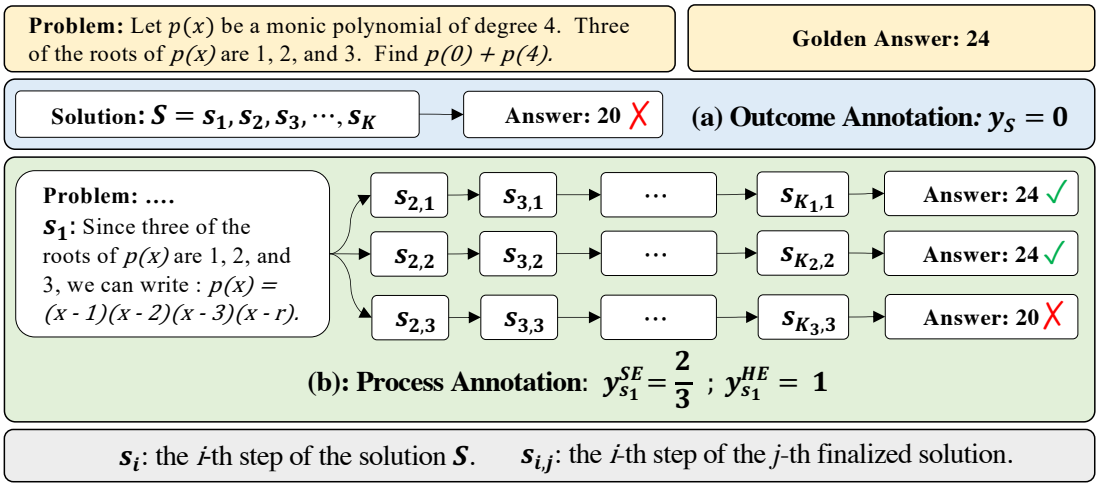

Figure 1: Comparison for previous automatic outcome annotation and our automatic process annotation. (a): automatic outcome annotation assigns a label to the entire solution S, dependent on the correctness of the answer; (b) automatic process annotation employs a 'completer' to finalize N reasoning processes (N=3 in this figure) for an intermediate step (s_1 in this figure), subsequently use hard estimation (HE) and soft estimation (SE) to annotate this step based on all decoded answers.

step that has the potential to deduce a well-founded result can be considered a good reasoning step. Analogous to ORM, this definition also introduces some degree of noise. Nevertheless, we find that it is beneficial for effectively training a good PRM.

3.3.2 Solution

Completion To quantify and estimate the *potential* for a give reasoning step s_i, as shown in Figure 1, we use a 'completer' to finalize N subsequent reasoning processes from this step: $\{(s_{i+1,j}, \cdots, s_{K_j,j}, a_j)\}_{j=1}^{N}$, where a_j and K_j are the decoded answer and the total number of steps for the j-th finalized solution, respectively. Then, we estimate the potential of this step based on the correctness of all decoded answers $A = \{a_j\}_{j=1}^{N}$.

Estimation In this paper, we use two methods to estimate the quality y_{s_i} for the step s_i, hard estimation (HE) and soft estimation (SE). HE supposes that a reasoning step is good as long as it can reach the correct answer a^*:

$$y_{s_i}^{HE} = \begin{cases} 1 & \exists a_j \in A, a_j = a^* \\ 0 & \text{Otherwise} \end{cases} \quad (3)$$

SE assumes the quality of a step as the frequency with which it reaches the correct answer:

$$y_{s_i}^{SE} = \frac{\sum_{j=1}^{N} \mathbb{I}(a_j = a^*)}{N}. \quad (4)$$

Once we gather the label of each step, we can train PRM with the cross-entropy loss. In conclusion, our automatic process annotation framework defines the quality of a step as its potential to deduce the correct answer and achieve the label of each step by completion and estimation.

3.4 Ranking for Verification

Following (Lightman et al., 2023), we use the minimum score across all steps to represent the final score of a solution assigned by PRM. We also explore the combination of self-consistency and reward models following (Li et al., 2023b). In this context, we initially classify solutions into distinct groups according to their final answers. Following that, we compute the aggregate score for each group. Formally, the final prediction answer based on N candidate solutions is:

$$a_{sc+rm} = \arg\max_{a} \sum_{i=1}^{N} \mathbb{I}(a_i = a) \cdot RM(p, S_i). \quad (5)$$

Where $RM(p, S_i)$ is the score of the i-th solution assigned by ORM or PRM for problem p.

3.5 Reinforcement Learning

Upon achieving PRM, we employ reinforcement learning to train LLMs. We implement Proximal Policy Optimization (PPO) in a step-by-step manner. This method differs from the conventional outcome RL with ORM, which only offers a reward at the end of the response (Ouyang et al., 2022). Conversely, process RL offers rewards at the end of each reasoning step. Formally, for the response

with n tokens, Outcome RL provides the reward at the last token:

$$r_t = \begin{cases} t \neq n & 0 \\ t = n & r_{\text{ORM}} \end{cases}, \tag{6}$$

while process RL provides rewards for each step:

$$r_t = \begin{cases} t \notin \mathbb{EOS} & 0 \\ t \in \mathbb{EOS} & r_{\text{PRM}_t} \end{cases}, \tag{7}$$

where $\mathbb{EOS}$ denotes the set of indices that correspond to the end of each step.

4 Experiments

Datasets We conduct our experiments using two widely used math reasoning datasets, GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021). For the GSM8K dataset, we leverage the whole test set in both verification and reinforcement learning scenarios. For the MATH dataset, in the verification scenario, due to the computation cost, we employ a subset MATH500 that is identical to the test set of Lightman et al. (2023). The subset consists of 500 representative problems, and we find that the subset evaluation produces similar results to the full-set evaluation. To assess different verification methods, we generate 256 candidate solutions for each test problem. We report the mean accuracy of 3 groups of sampling results. In the reinforcement learning scenario, we use the whole test set to evaluate the model performance. We train LLMs with MetaMATH (Yu et al., 2023b).

Parameter Setting Our experiments are based on a series of large language models, LLaMA2-7B/13B/70B (Touvron et al., 2023), LLemma-7B/34B (Azerbayev et al., 2023), Mistral-7B (Jiang et al., 2023) and DeepSeek-67B (DeepSeek, 2023). We train the generator and completer for 3 epochs on MetaMATH. We train the Mistral-7B with a learning rate of 5e-6. For other models, The learning rates are set to 2e-5, 1e-5, and 6e-6 for the 7B/13B, 34B, and 67B/70B LLMs, respectively. To construct the training dataset of ORM and PRM, we train 7B and 13B models for a single epoch on the GSM8K and MATH training sets. Subsequently, we sample 15 solutions per problem from each model for the training set. Following this, we eliminate duplicate solutions and annotate the solutions at each step. We use LLemma-7B as the completer with the decoded number N=8. Consequently, we obtain around 170k solutions for GSM8K and 270k solutions for MATH. For verification, we choose LLaMA2-70B and LLemma-34B as the base models to train reward models for GSM8K and MATH, respectively. For reinforcement learning, we choose Mistral-7B as the base model to train reward models and use it to supervise LLama2-7B and Mistral-7B generators. The reward model is trained in 1 epoch with a learning rate 1e-6. For the sake of convenience, we train the PRM using the hard estimation version because it allows us to utilize a standard language modeling pipeline by selecting two special tokens to represent 'has potential' and 'no potential' labels, thereby eliminating the need for any specific model adjustments. In reinforcement learning, the learning rate is 4e-7 and 1e-7 for LLaMA2-7B and Mistral-7B, respectively. The Kullback-Leibler coefficient is set to 0.04. We implement a cosine learning rate scheduler, employing a minimal learning rate set to 1e-8. We use HAI-LLM (High-flyer, 2023) to train all models with the max sequence length of 512.

Baselines and Metrics In the verification scenario, following (Lightman et al., 2023), we evaluate the performance of our reward model by comparing it against the Self-consistency (majority voting) and outcome reward model. The accuracy of the best-of-N solution is utilized as the evaluation metric. For PRM, the minimum score across all steps is adopted to represent the final score of a solution. In the reinforcement scenario, we compare our step-by-step supervision with the outcome supervision provided by ORM, and Rejective Sampling Fine-tuning (RFT) (Yuan et al., 2023), we sample 8 responses for each question in MetaMATH for RFT. We use the accuracy of LLMs' greedy decoding output to assess the performance.

4.1 Main Results

MATH-SHEPHERD as verifier Table 1 presents the performance comparison of various methods on GSM8K and MATH. We find that: 1) As the verifier, MATH-SHEPHERD consistently outperforms self-consistency and ORM on two datasets with all generators. Specifically, enhanced by MATH-SHEPHERD, DeepSeek-67B achieves 93.3% and 48.1% accuracy on GSM8K and MATH; 2) In comparison to GSM8K, PRM achieves a greater advantage over ORM on the more challenging MATH dataset; This outcome aligns with the findings in (Uesato et al., 2022) and (Lightman et al., 2023).

Models	Verifiers	GSM8K	MATH500
LLaMA2-70B: MetaMATH	Self-Consistency	88.0	39.4
	ORM	91.8	40.4
	Self-Consistency + ORM	92.0	42.0
	MATH-SHEPHERD (Ours)	93.2	44.5
	Self-Consistency + MATH-SHEPHERD (Ours)	92.4	45.2
LLemma-34B: MetaMATH	Self-Consistency	82.6	44.2
	ORM	90.0	43.7
	Self-Consistency + ORM	89.6	45.4
	MATH-SHEPHERD (Ours)	90.9	46.0
	Self-Consistency + MATH-SHEPHERD (Ours)	89.7	47.3
DeepSeek-67B: MetaMATH	Self-Consistency	88.2	45.4
	ORM	92.6	45.3
	Self-Consistency + ORM	92.4	47.0
	MATH-SHEPHERD (Ours)	**93.3**	47.0
	Self-Consistency + MATH-SHEPHERD (Ours)	92.5	**48.1**

Table 1: Performances of different LLMs on GSM8K and MATH with different verification strategies. The reward models are trained based on LLama2-70B and LLemma-34B on GSM8K and MATH, respectively. The verification is based on 256 outputs. We report the mean accuracy of 3 groups of sampling results.

Models	Verifiers	GSM8K	MATH500
Mistral-7B: MetaMATH	Self-Consistency	83.9	35.1
	ORM	86.2	36.4
	Self-Consistency + ORM	86.6	38.0
	MATH-SHEPHERD (Ours)	87.1	37.3
	Self-Consistency + MATH-SHEPHERD (Ours)	86.3	38.3
Mistral-7B: MetaMATH +Process RL (Ours)	Self-Consistency	87.4	42.3
	ORM	87.6	41.3
	Self-Consistency + ORM	89.0	43.1
	MATH-SHEPHERD (Ours)	88.4	41.1
	Self-Consistency + MATH-SHEPHERD (Ours)	**89.1**	**43.5**

Table 2: Results of reinforcement learning and verification combination. The reward models are trained based on Mistral-7B. The verification is based on 256 outputs. We report the mean accuracy of 3 groups of sampling results.

The former discovers that PRM and ORM yield similar results on GSM8K, whereas the latter shows that PRM significantly outperforms ORM on the MATH dataset. This could be attributed to the relative simplicity of the GSM8K dataset compared to MATH, i.e., the GSM8K dataset necessitates fewer steps for problem-solving. As a result, ORM operates efficiently when handling this particular dataset; 3) In GSM8K, when combined with self-consistency, there's a drop in performance, whereas in MATH, performance improves. These results indicate that if the reward model is sufficiently powerful for a task, combining it with self-consistency may harm the verification performance.

MATH-SHEPHERD as reward model on reinforcement learning Table 3 presents the performance of different LLMs with greedy decoding outputs. As is shown: 1) RL with process supervision significantly improves the performance of two supervised fine-tuned models. For example, Mistral-7B with Process RL achieves 84.1% and 33.0% on the GSM8K and MATH datasets, respectively; 2) RFT only slightly improves the model performance, we believe this is because MetaMATH already has conducted some data augmentation strategies like RFT; 3) Outcome RL can also enhance the model performance. However, it does not perform as well as the Process RL with MATH-SHEPHERD, demonstrating the potential of our method.

MATH-SHEPHERD as both reward models and verifiers We also combine RL and verification. As shown in Table 2: 1) RL and verification are complementary. For example, in MATH, Mistral-7B utilizing process RL outperforms supervised fine-tuning Mistral-7B 7.2% accuracy with self-consistency as the verifier; The performance gap

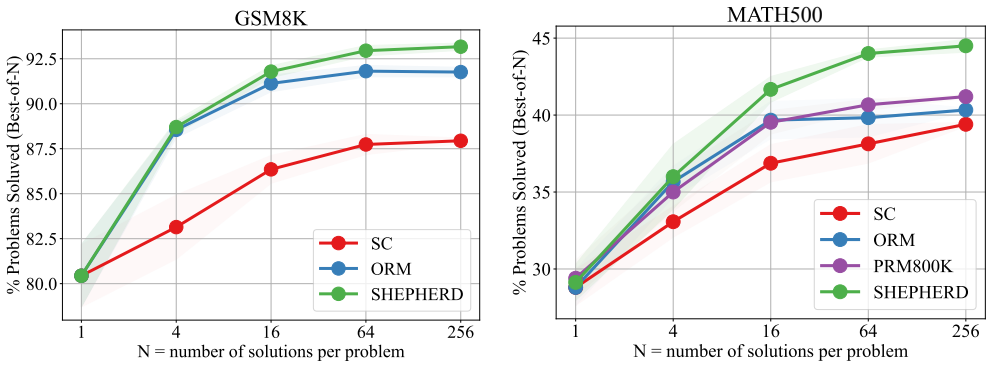

Figure 2: Performance of LLaMA2-70B using different verification strategies across different numbers of solution candidates on GSM8K and MATH.

Models	GSM8K	MATH
LLaMA2-7B: MetaMATH	66.6	19.2
+ RFT	68.5	19.9
+ Outcome RL	70.8	20.8
+ Process RL	73.2	21.6
Mistral-7B: MetaMATH	77.9	28.6
+ RFT	79.0	29.9
+ Outcome RL	81.8	31.3
+ Process RL	**84.1**	**33.0**

Table 3: Performances of different models with greedy decoding. We use the questions in MetaMATH for RFT and RL. Both LLaMA2-7B and Mistral-7B are supervised by Mistral-7B-ORM and -MATH-SHEPHERD.

is even larger than that of greedy decoding results, i.e., 4.4%; 2) after RL, the vanilla verification methods with only reward models are inferior to self-consistency, we think the reason is that the initial reward model is not sufficient to supervise the RL-enhanced model. These results can show the potential of iterative RL, which we leave for future work.

5 Analysis

5.1 Performance with Different Number of Candidate Solutions

Figure 2 illustrates the performance comparison of various strategies when applied to different numbers of candidates ranging from 1 to 256 on two benchmarks. The key observations are as follows: 1) PRM exhibits consistent superior performance when compared to both ORM and majority voting, with the degree of this superiority becoming more pronounced as N escalates. 2) In MATH, our automatically annotated datasets outperform the human-annotated PRM800K (Lightman et al., 2023). We ascribe this superiority to the distribution gap and the data quantity. Specifically, PRM800K is annotated based on the output from GPT-4, and consequently, a discrepancy arises for the output of open-source LLaMA models fine-tuned on MetaMATH. Furthermore, when considering the quantity of data, our automated reward model data exhibits both high scalability and a reduced labeling cost. Consequently, our dataset is four times larger than that provided in PRM800K. Overall, these results further underscore the effectiveness and potential of our method. We also explore the performance of different verification strategies on different sizes of generators and verifiers from 7B to 70B. Please refer to Appendix A for details.

5.2 Quality of Automatic Process Annotations

In this section, we explore the quality of our automatic PRM dataset. To achieve this, we manually annotate 160 steps sampled from the training set of GSM8K and use different completers to infer from each step to achieve their label. We find that:

Automatic process annotation exhibits satisfactory quality. Figure 3(a) demonstrates that utilizing LLaMA2-70B trained on MetaMATH as the completer, the accuracy of the hard estimation (HE) reaches 86% when N equals 4. This suggests that our automatically constructed dataset is of high quality. However, we observed a decline in the accuracy of the constructed dataset with further increases in N. Our analysis indicates that larger values for N may lead to false positives.

Figure 3(b) shows the cross-entropy loss between SE and HE labels compared to the human-annotated distribution: as N increases, SE progres-

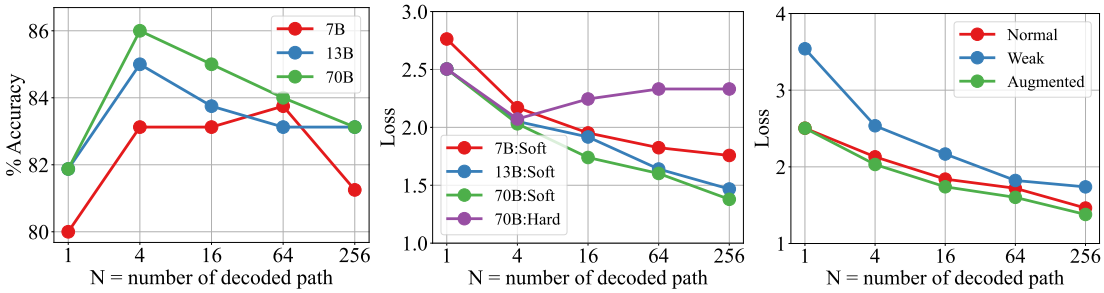

Figure 3: Quality of process annotation on GSM8K. (a): Accuracy of the process annotation using different completer; (b): Loss of the process annotation using different completer; (c): Loss of the process annotation using the same completer with different training data.

Methods	Models	Acc.
DIVERSE-NLI	DeBERTa	61.3
DIVERSE-NLI	LLaMA2-13B	75.6
DIVERSE-Rule	-	75.0
MATH-SHEPHERD	LLaMA2-13B (N = 4)	85.0

Table 4: Performance of MATH-SHEPHERD and NLI/Rule-based annotation methods (Li et al., 2023b).

sively aligns closer to the standard distribution, in contrast to HE which does not exhibit similar behavior. It is essential to note that at N=4, HE achieves an accuracy of 86%. We can theoretically attain higher quality data exceeding 86% accuracy by utilizing SE. However, we discovered that the performance of the verifier exhibits no substantial divergence whether trained with either SE or HE. This may be attributable to the already high-quality annotations provided by HE.

Furthermore, we also delve into other automatic process annotation methodologies. For instance, (Li et al., 2023b) employs a natural language inference (NLI) model and a string match rule to annotate a given step. The NLI-based method annotates a step as correct if it is entailment with any step in the reference solutions. The Rule-based method annotates a step as correct if its support number precisely matches that of any steps in the reference solutions. As demonstrated in Table 4, our annotation strategy exhibits substantial superiority over the two approaches.

The ability of the LLM completer plays an important role in the data quality. We employ a completer to finalize multiple subsequent reasoning processes for a given step. Therefore, we investigate the impact of the LLM completer. Figure 3(b) presents the cross-entropy loss across diverse completers trained on MetaMath. The results indicate that a larger completer is adept at generating superior-quality datasets. Figure 3(c) depicts the cross-entropy loss of LLaMA2-70B trained with different datasets. 'Normal' denotes the original GSM8K training dataset; 'Weak' refers to the Normal set excluding examples whose questions are in our 160 evaluation set; while 'Augmented' symbolizes MetaMath, an augmented version of the Normal set. The findings suggest that high-quality training sets allow the model to operate more proficiently as a completer. Importantly, the 'Weak' set exhibits a markedly larger loss than other datasets. This insight drives us to infer that LLMs should acquire the questions in advance to enhance their performance as completers. We can also conjecture that a stronger foundational model, coupled with superior training data, could further enhance the quality of automatic annotation.

5.3 Influence of the Number of Data

We delve deeper into the analysis of PRM and ORM by utilizing varying quantities of training data. As depicted in Figure 4(a), it is clear that PRM exhibits superior data efficiency. Specifically, it outperforms ORM by approximately 4% accuracy when applying a modestly sized training dataset (i.e., 10k instances). Furthermore, PRM seems to have a higher potential ceiling than ORM. These observations highlight the efficacy of PRM for verification purposes.

5.4 Out-of-distribution Performance

To further demonstrate the effectiveness of our method, we conduct an out-of-distribution evaluation on the Hungarian national final exam[1], which

[1] https://huggingface.co/datasets/keirp/hungarian_national_hs_finals_exam

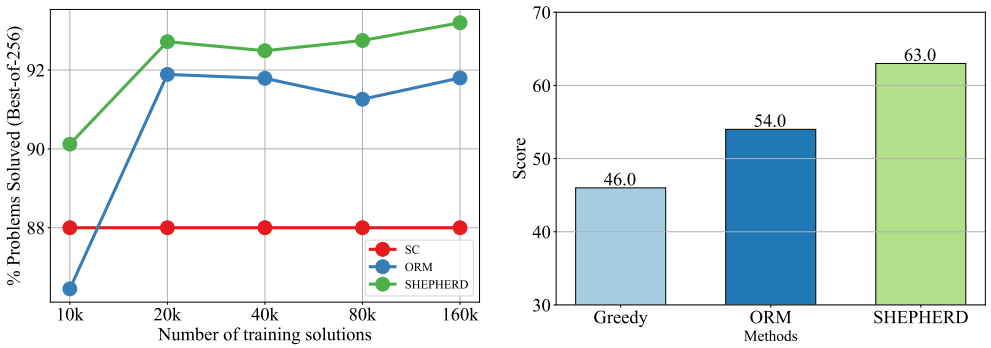

Figure 4: (a): Performance of different reward models using different numbers of training data; (b) performance of different verification strategies on the out-of-distribution Hungarian national exam.

consists of 33 questions. The total score of these questions is 100. We use the LLemma-34B trained on MetaMATH to serve as the generator and generate 256 candidate solutions for each question. We use LLemma-34B-ORM and LLemma-34B-PRM to select the solution for each question. As shown in Figure 4(b): 1) both LLemma-34B-ORM and LLemma-34B-PRM outperform the origin LLemma-34B, showing the reward model can generalize to other domains; 2) PRM outperforms ORM 9 scores, further demonstrating the superiority of PRM. We also conduct a case study to intuitively demonstrate the effectiveness of MATH-SHEPHERD. Please refer to Appendix B for details.

6 Conclusion

In this paper, we introduce a process-oriented math verifier called MATH-SHEPHERD, which assigns a reward score to each step of the LLM's outputs on math problems. The training of MATH-SHEPHERD is achieved using automatically constructed process-wise supervision data, thereby eradicating the necessity for labor-intensive human annotation. Remarkably, this automatic methodology correlates strongly with human annotations. Extensive experiments in both verification and reinforcement learning scenarios demonstrate the effectiveness of our method.

Limitations

Our paper has some limitations, which we leave for future work:

The computational cost of the completion process. To determine the label of each reasoning step, we utilize a 'completer' to decode N subsequent reasoning processes. We observe that as N increases, so does the quality of automatic annotations. However, this completion process demands a lot of computing resources, potentially imposing a limitation on the usage of our method. Despite this limitation, the cost remains significantly lower than human annotation. Furthermore, we are optimistic that advancements in efficient inference techniques such as speculative decoding (Xia et al., 2022; Leviathan et al., 2023) and vLLM (Kwon et al., 2023) could mitigate this limitation.

The automatic process annotation consists of noise. Similar to the automatic outcome annotation, our automatic process annotation also has noise. Despite this, our experiments verify the efficacy of our method for training a PRM. In particular, the PRM trained on our dataset outperforms the human-annotated PRM800K dataset. However, a noticeable gap remains between PRM800K and the candidate responses generated by the open-source models utilized in this study, which may result in the invalidation of PRM800K. As a result, the impact of this potential noise on PRM performance is still undetermined. A comprehensive comparison between human and automated annotations is envisaged for future studies. Furthermore, we assert that integrating human and automated process annotations could play a vital role in constructing robust and efficient process supervision.

Acknowledgements

This paper is supported by the National Key Research and Development Program of China 2020AAA0106700. The contact author is Zhifang Sui.

References

Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q Jiang, Jia Deng, Stella Biderman, and Sean Welleck. Llemma: An open language model for mathematics. *arXiv preprint arXiv:2310.10631*, 2023.

Zhen Bi, Ningyu Zhang, Yinuo Jiang, Shumin Deng, Guozhou Zheng, and Huajun Chen. When do program-of-thoughts work for reasoning? *arXiv preprint arXiv:2308.15452*, 2023.

Liang Chen, Yichi Zhang, Shuhuai Ren, Haozhe Zhao, Zefan Cai, Yuchi Wang, Peiyi Wang, Tianyu Liu, and Baobao Chang. Towards end-to-end embodied decision making via multi-modal large language model: Explorations with gpt4-vision and beyond. *arXiv preprint arXiv:2310.02071*, 2023.

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

Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pp. 72–83. Springer, 2006.

DeepSeek. Deepseek llm: Let there be answers. https://github.com/deepseek-ai/DeepSeek-LLM, 2023.

Yao Fu, Hao Peng, Ashish Sabharwal, Peter Clark, and Tushar Khot. Complexity-based prompting for multi-step reasoning. *arXiv preprint arXiv:2210.00720*, 2022.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Minlie Huang, Nan Duan, Weizhu Chen, et al. Tora: A tool-integrated reasoning agent for mathematical problem solving. *arXiv preprint arXiv:2309.17452*, 2023.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

High-flyer. Hai-llm: Efficient and lightweight training tool for large models, 2023. URL https://www.high-flyer.cn/en/blog/hai-llm.

Jie Huang and Kevin Chen-Chuan Chang. Towards reasoning in large language models: A survey. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 1049–1065, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.67. URL https://aclanthology.org/2023.findings-acl.67.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*, 2023.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

Jean Kaddour, Joshua Harris, Maximilian Mozes, Herbie Bradley, Roberta Raileanu, and Robert McHardy. Challenges and applications of large language models. *arXiv preprint arXiv:2307.10169*, 2023.

Muhammad Khalifa, Lajanugen Logeswaran, Moontae Lee, Honglak Lee, and Lu Wang. Grace: Discriminator-guided chain-of-thought reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 15299–15328, 2023.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pp. 282–293. Springer, 2006.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023.

Lei Li, Yuwei Yin, Shicheng Li, Liang Chen, Peiyi Wang, Shuhuai Ren, Mukai Li, Yazheng Yang, Jingjing Xu, Xu Sun, et al. M3it: A large-scale dataset towards multi-modal multilingual instruction tuning. *arXiv preprint arXiv:2306.04387*, 2023a.

Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. Making language models better reasoners with step-aware verifier. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5315–5333, Toronto, Canada, July 2023b. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.291. URL https://aclanthology.org/2023.acl-long.291.

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

Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. *arXiv preprint arXiv:2308.09583*, 2023.

Qianli Ma, Haotian Zhou, Tingkai Liu, Jianbo Yuan, Pengfei Liu, Yang You, and Hongxia Yang. Let's reward step by step: Step-level reward model as the navigators for reasoning. *arXiv preprint arXiv:2310.10080*, 2023.

OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023. doi: 10.48550/arXiv.2303.08774. URL https://doi.org/10.48550/arXiv.2303.08774.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

Sarah Pan, Vladislav Lialin, Sherin Muckatira, and Anna Rumshisky. Let's reinforce step by step. *arXiv preprint arXiv:2311.05821*, 2023.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–22, 2023.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Yifan Song, Weimin Xiong, Dawei Zhu, Cheng Li, Ke Wang, Ye Tian, and Sujian Li. Restgpt: Connecting large language models with real-world applications via restful apis. corr, abs/2306.06624, 2023. doi: 10.48550. *arXiv preprint arXiv.2306.06624*.

Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk. Monte carlo tree search: A review of recent modifications and applications. *Artificial Intelligence Review*, 56(3):2497–2562, 2023.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023a.

Peiyi Wang, Lei Li, Liang Chen, Feifan Song, Binghuai Lin, Yunbo Cao, Tianyu Liu, and Zhifang Sui. Making large language models better reasoners with alignment. *arXiv preprint arXiv:2309.02144*, 2023b.

Peiyi Wang, Lei Li, Liang Chen, Dawei Zhu, Binghuai Lin, Yunbo Cao, Qi Liu, Tianyu Liu, and Zhifang Sui. Large language models are not fair evaluators. *arXiv preprint arXiv:2305.17926*, 2023c.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023d. URL https://openreview.net/pdf?id=1PL1NIMMrw.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, 2022.

Zeqiu Wu, Yushi Hu, Weijia Shi, Nouha Dziri, Alane Suhr, Prithviraj Ammanabrolu, Noah A Smith, Mari Ostendorf, and Hannaneh Hajishirzi. Fine-grained human feedback gives better rewards for language model training. *arXiv preprint arXiv:2306.01693*, 2023.

Heming Xia, Tao Ge, Furu Wei, and Zhifang Sui. Lossless speedup of autoregressive translation with generalized aggressive decoding. *arXiv preprint arXiv:2203.16487*, 2022.

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

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*, 2023b.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Chuanqi Tan, and Chang Zhou. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023.

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. Mammoth: Building math generalist models through hybrid instruction tuning. *arXiv preprint arXiv:2309.05653*, 2023.

Yifan Zhang, Jingqin Yang, Yang Yuan, and Andrew Chi-Chih Yao. Cumulative reasoning with large language models. *arXiv preprint arXiv:2308.04371*, 2023.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *arXiv preprint arXiv:2306.05685*, 2023.

Xinyu Zhu, Junjie Wang, Lin Zhang, Yuxiang Zhang, Yongfeng Huang, Ruyi Gan, Jiaxing Zhang, and Yujiu Yang. Solving math word problems via cooperative reasoning induced language models. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4471–4485, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.245. URL https://aclanthology.org/2023.acl-long.245.

A Influence of the Model Size for Verification

To conduct an exhaustive evaluation of MATH-SHEPHERD's effectiveness, we performed a diverse range of experiments using model sizes 7B, 13B, and 70B. Figures 5(a), 5(b), and 2(a) display the results from the 7B, 13B, and 70B generators paired with equal-sized reward models, respectively. It becomes evident that PRM exhibits superiority over self-consistency and ORM across all sizes of base models. Moreover, bigger reward models prove to be more robust; for instance, the accuracy of the 70B reward models escalates as the number of candidate solutions rises, while the 7B reward models show a decreasing trend.

Figure 5(c) and 5(d) presents the performance of 7B and 70B generators interfaced with different-sized reward models. The findings illustrate that utilizing a larger reward model to validate the output of a smaller generator significantly enhances performance. Conversely, when a smaller reward model is employed to validate the output of a larger generator, the verification process adversely impacts the model's performance compared to SC. These results substantiate that we should utilize a more potent reward model for validating or supervising the generator.

B Case Study

As outlined in Table 5, when presented with a question from the Hungarian national final exam, our MATH-SHEPHERD accurately selected the correct solution from a pool of 256 potential solutions, which ORM failed. Moreover, MATH-SHEPHERD displayed superior discernment by precisely identifying incorrect steps within the solutions selected by ORM. Notably, it recognized errors in Step 2, Step 6, and Step 9 and so on, and subsequently assigned them lower scores relative to those for steps present in the correct solutions.

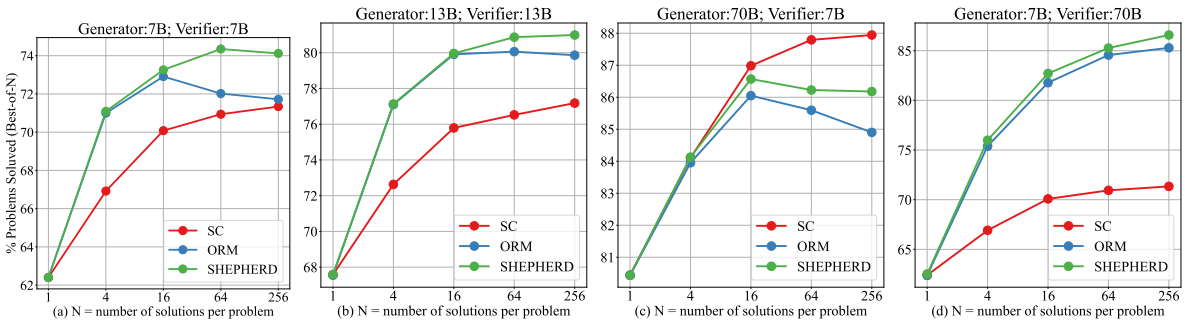

Figure 5: Performance of different verification strategies on different sizes of generators and verifiers.

Question	*The first term of an arithmetic sequence is 18. The sum of the first six terms of the sequence is equal to the sum of the first seven terms. Show that the sum of the first 13 terms of the sequence is 0 and calculate the 13th term of the sequence.*
MATH-SHEPHERD	Step 1: Let the common difference of the arithmetic sequence be d. MATH-SHEPHERD: 0.99. Step 2: The sum of the first six terms of the sequence is $18 + (18 + d) + (18 + 2d) + (18 + 3d) + (18 + 4d) + (18 + 5d) = 108 + 30d$. MATH-SHEPHERD: 0.85. ..... Step 6: Therefore, $d = -3$. MATH-SHEPHERD: 0.85. ..... Step 8: The 13th term of the sequence is $18 - 36 = -18$. MATH-SHEPHERD: 0.93. ORM Score: 0.73.
ORM	Step 1: Let the common difference of the arithmetic sequence be d. MATH-SHEPHERD: 0.99. Step 2: The sum of the first six terms is $18 + (18 + d) + (18 + 2d) + (18 + 3d) + (18 + 4d) + (18 + 5d) =$ 90 + $15d$. MATH-SHEPHERD: 0.53. ..... Step 6: Dividing by -6, we find that $d = -2$. MATH-SHEPHERD: 0.38. ..... Step 9: The 13th term of the sequence is $18 - 26 = -8$. MATH-SHEPHERD: 0.38. ORM Score: 0.84.

Table 5: A case study from the Hungarian national exam. Red text denotes the mistake that ORM fails to detect.