SOCRATIC-PRMBENCH: Benchmarking Process Reward Models with Systematic Reasoning Patterns

Anonymous ACL submission

Abstract

Process Reward Models (PRMs) are crucial in complex reasoning and problem-solving tasks (e.g., LLM agents with long-horizon decision-making) by verifying the correctness of each intermediate reasoning step. In real-world scenarios, LLMs may apply various reasoning patterns (e.g., decomposition) to solve a problem, potentially suffering from errors under various reasoning patterns. Therefore, PRMs are required to identify errors under various reasoning patterns during the reasoning process. However, existing benchmarks mainly focus on evaluating PRMs with stepwise correctness, ignoring a systematic evaluation of PRMs under various reasoning patterns. To mitigate this gap, we introduce SOCRATIC-PRMBENCH, a new benchmark to evaluate PRMs systematically under six reasoning patterns, including *Transformation*, *Decomposition*, *Regather*, *Deduction*, *Verification*, and *Integration*. SOCRATIC-PRMBENCH comprises 2995 reasoning paths with flaws within the aforementioned six reasoning patterns. Through our experiments on both PRMs and LLMs prompted as critic models, we identify notable deficiencies in existing PRMs. These observations underscore the significant weakness of current PRMs in conducting evaluations on reasoning steps under various reasoning patterns. We hope SOCRATIC-PRMBENCH can serve as a comprehensive testbed for systematic evaluation of PRMs under diverse reasoning patterns and pave the way for future development of PRMs.

1 Introduction

Large Language Models (LLMs) (OpenAI, 2024b; DeepSeek-AI, 2025; Team, 2024b) augmented by methodologies like Reinforcement Learning with Verifialble Rewards (RLVR) (Trung et al., 2024; Shao et al., 2024) and Test-Time Scaling (Snell et al., 2025; Bansal et al., 2025), have demonstrated significant capabilities in complex reasoning and decision-making tasks. Process Reward Models (PRMs) (Lightman et al., 2024; Wang et al., 2023; Zhang et al., 2025) play a crucial role in these advancements, especially for LLM agents which involve long-horizon decision-making steps (Choudhury, 2025; Ma et al., 2025; Xiong et al., 2025). By providing step-level rewards during the reasoning process, PRMs offer more accurate and denser reward signals, which in turn guide the optimization of LLMs and the exploration of reasoning trajectories (Tie et al., 2025; Ji et al., 2025).

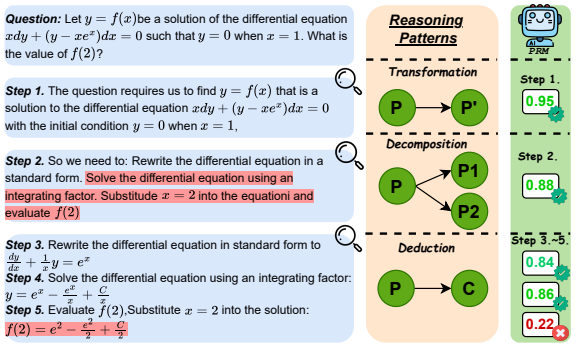

Figure 1: (Left): Given a question, the reasoning step 2 and 5 contain errors. (Medium): Each step applys a specific reasoning pattern. (Right): The process reward model successfully detects the error of *Deduction* pattern but fails with the *Decomposition* reasoning pattern.

However, the diverse reasoning patterns applied by LLMs during reasoning process (Dong et al., 2023; Li et al., 2024) pose a challenge for PRMs in consistently providing accurate rewards. Figure 1 illustrates such a scenario: according to the thoery of ancient Greek philosopher Socrate (Dong et al., 2023; Qi et al., 2023), the reasoning pattern for Step 1 is '*Transformation*', for Step 2 '*Decomposition*', and for Steps 3-5 '*Deduction*'. Although the existing PRM identifies the error in Step 5 (*Deduction* pattern), it does not detect the fundamental cause of this error from the *Decomposition* pattern. Specifically, in Step 2, the omission of substituting a point in the solution of the differential equation to calculate the constant C, causes C to remain

	PRM Benchmarks?	Error Type Detection?	Fine-grained Classes	Reasoning Patterns†	Annotator	Test Case Size	Average Steps
RMBench (Liu et al., 2025)	✗	✗	1	1	Synthetic + Human	1,327	-
CriticBench (Lin et al., 2024)	✗	✗	1	1	-	-	-
MathCheck-GSM (Zhou et al., 2025)	✗	✗	1	1	Synthetic	516	-
ProcessBench (Zheng et al., 2024)	✓	✗	1	1	Human	3,400	7.1
PRMBench (Song et al., 2025)	✓	✓	9	1	Synthetic + Human	6,216	13.4
SOCRATIC-PRMBENCH	✓	✓	20	6	Synthetic + Human	2995	8.7

Table 1: Comparison between our proposed SOCRATIC-PRMBENCH and other benchmarks or datasets for reward model evaluation. †: the number of reasoning patterns covered within the benchmark.

undetermined throughout the subsequent reasoning process, resulting in a flawed final answer. This observation indicates the unreliability of current PRMs towards diverse reasoning patterns.

For a comprehensive assessment of PRMs' error detection capabilities across various reasoning patterns, we introduce SOCRATIC-PRMBENCH, a systematic and fine-grained benchmark. In contrast to prior benchmarks with limited systematic evaluation (Zheng et al., 2024; Song et al., 2025), inspired by the ancient Greek philosopher Socrates, we design to evaluate PRMs' proficiency in detecting errors across 6 reasoning patterns: *Transformation*, *Decomposition*, *Regather*, *Deduction*, *Verification*, and *Integration*. Specifically, SOCRATIC-PRMBENCH comprises 2995 reasoning paths, with flaws categorized into six primary categories by reasoning pattern and 20 sub-categories of fine-grained error types. The data annotation process for SOCRATIC-PRMBENCH is fully automated using LLMs, thereby obviating the need for extensive human labor. We ensure the difficulty of the data through rule-based filtering and guarantee its quality through manual expert review.

We conducted extensive experiments on a wide range of models, including open-source PRMs, and a series of general-purpose and reasoning-specialized LLMs. The findings reveal considerable scope for improvement in current PRMs. Notably, Qwen2.5-Math-PRM, the highest-performing PRM, attained a mere 68.0 overall score. Through detailed analytical experiments, we identified substantial disparities in the error detection capabilities of current PRMs across different reasoning patterns, alongside evident latency in indentifying error steps and significant bias of reward generation. By leveraging SOCRATIC-PRMBENCH for evaluation, we offer a pathway to comprehensively assess PRMs from the perspective of reasoning patterns. This can be potentially helpful for mitigating the risk of reward hacking in future PRM development. In general, our contributions are summarized as follows:

- We propose SOCRATIC-PRMBENCH, the first systematic PRM benchmark from reasoning pattern perspective, comprising 2995 samples for a comprehensive and fine-grained evaluation on process reward models.
- Based on ancient Greek logic theory (Qi et al., 2023), SOCRATIC-PRMBENCH covers 6 carefully designed reasoning patterns, including *Transformation*, *Decomposition*, *Regather*, *Deduction*, *Verification*, and *Integration*, with 20 sub-categories of fine-grained error types. This systematic and granular evaluation framework enables a comprehensive assessment of PRMs and facilitates the identification of their potential shortcomings.
- We perform extensive experiments on a wide range of SOTA PRMs and LLMs with SOCRATIC-PRMBENCH. Our results reveal essential limitations in current PRMs and offer insights for future progress in this area.

2 Related Work

Process Reward Models Process reward models (PRMs) have demonstrated their superiority over outcome reward models (ORMs) (Zhang et al., 2024; Ankner et al., 2024) by providing more accurate and dense reward signals for intermediate reasoning steps. As a result, the development of PRMs is gaining increasing attention. Lightman et al. (2024) contributes a manually annoted dataset for PRM training, Wang et al. (2024) propose an automatic step-level labeling method with Monte Carlo estimation. Moreover, Dong et al. (2024); Zhao et al. (2025) forms process reward modeling as generation task and improve generative capabilities of PRMs using CoT reasoning. In contrast to the flourish of PRMs' training, PRMs' evaluation remaines comparatively underdeveloped. To remedy this imbalance, we present SCORATIC-PRMBENCH, a novel benchmark for PRMs' evaluation.

Reward Model Benchmarks Reward benchmarks are crucial for evaluating reward models, as they provide a direct and quantifiable measure. Despite the emergence of numerous benchmarks (Liu et al., 2025; Lin et al., 2024; Lambert et al., 2024), they are are primarily designed to evaluate ORMs, without any step-level annotations. Zheng et al. (2024); Song et al. (2025) annotate step-level labels using LLMs and human experts to create benchmarks for PRMs. However, their evaluation are not systematic and ignore the need to evaluate PRMs' error detection capabilities towards diverse reasoning patterns (Dong et al., 2023; Li et al., 2024). To address this gap, we propose SOCRATIC-PRMBENCH, a systematic and granular benchmark to provide a comprehensive assessment of PRMs from the perspective of reasoning patterns. A comparison between our SOCRATIC-PRMBENCH and existing reward model benchmarks is summarized in Table 1.

3 Socratic-PRMBench

3.1 Reasoning Patterns

The design of the reasoning patterns in SOCRATIC-PRMBENCHMARK is inspired by the logical theories of the ancient Greek philosopher Socrates. As Socrates once stated, "I cannot teach anybody anything. I can only make them think." Following this philosophical wisdom, we categorize reasoning into six atomic reasoning patterns, within these six reasoning patterns, we systematically design a total of 20 types of reasoning errors. The atomic reasoning patterns and the fine-grained categories of error types under Socrates' logical framework are illustrated in Figure 2.

Transformation transforms the problem into a homogeneous or similar problem, or abstract the problem. It usually explains the problem from a problem-solving perspective, aiming at gaining a more comprehensive and clear understanding of the problem. Specifically, the *Transformation* evaluation category can be divided into two sub-categories: ***Transformation Inconsistency*** and ***Transformation Counter-Factuality***. For a *Transformation* step $P \rightarrow P'$, Transformation Inconsistency refers that P' lacks consistency in logic, semantics, or understanding with P. Transformation Counter-Factuality refers to including factual error that against ground truth G in P'.

Decomposition breaks the problem into manageable subproblems, or makes a plan for reasoning steps, resolving the main problem by tackling each sub-problem. Specifically, the *Decomposition* evaluation category can be divided into three sub-categories: ***Decomposition Unsoundness***, ***Decomposition Redundancy***, and ***Decomposition Incompleteness***. For a *Decomposition* step $P \rightarrow \{P_1, P_2, ..., P_n\}$, each of the three sub-categories represents a distinct type of error in sub-problem P_i, which can be incorrect caused by logical inequality, missing important sub-problems and conditions, or including redundant sub-problems and constrains.

Regather collects key information from the input relevant to solving the problem and identifies crucial principles, and other concepts related to solving the problem. Specifically, the *Regather* evaluation category can be divided into three sub-categories: ***Regather Imprecision***, ***Regather Redundancy***, and ***Regather Incompleteness***. For a *Regather* step $P \rightarrow \{Q_1, Q_2, ..., Q_n\}$, Regather imprecision refers to collecting a Q_i with misinformation, misusing definations that are not suitable for solving the problem P. Regather Redundancy gathers redundant or unrelevant information not related with P. Regather Incompleteness refers to the absence of core definations, critical principles and concepts.

Deduction derives a conclusion for a given premise directly. Specifically, the *Deduction* evaluation category can be divided into six sub-categories: ***Premise Unsoundness***, ***Premise Incompleteness***, ***Premise Redundancy***, ***Conclusion Invalidity***, ***Conclusion Inconsistency*** and ***Conclusion Counter-Factuality***. For a *Deduction* step $P \rightarrow C$, the first three sub-categories arise from the premise and include: (1) starting deduction resoning from an unreasonable or incorrect premise, (2) introducing redundant assumptions into the premise, and (3) omitting key conditions and constraints. The remaining three sub-categories originate from the conclusion and include: (1) deriving an invalid conclusion from correct premises, (2) deriving a conclusion that contradicts a previous conclusion, and (3) deriving a conclusion that is inconsistent with known ground truth.

Verification examines reasoning steps in terms of factual accuracy, logical consistency, etc, detecting potential errors and refining them iteratively. Specifically, the *Verification* evaluation category can be divided into two sub-categories: ***Detection Error*** and ***Correction Error***. The former refers to failing to identify an incorrect conclusion C,

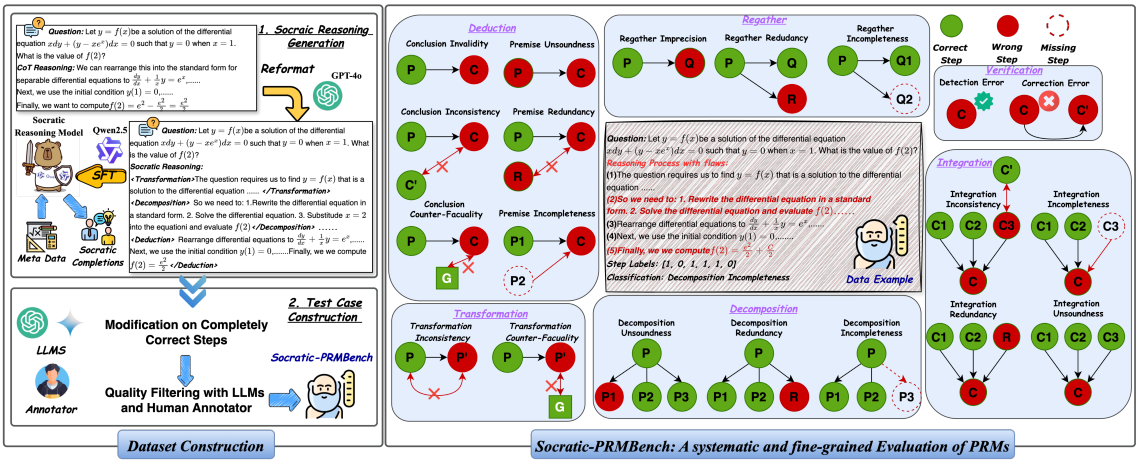

Figure 2: An overview of our SOCRATIC-PRMBENCH. The left part illustrates our dataset constuction procedure. The right part illustrates the 6 reasoning patterns and 20 sub-categories of fine-grained error types. We use P and C to represent (sub)problems and conclusions, respectively. We use Q, R, G to represent gathered information, redundant contents, and ground truth.

The latter, however, involves recognizing the initial error in C but introducing a new error during the attempted correction, leading to a different, incorrect conclusion C'.

Integration summarizes concluded conclusions to derive a new conclusion, integrating all current reasoning processes to form the final conclusion. Specifically, the *Integration* evaluation category can be divided into four sub-categories: ***Integration Inconsistency***, ***Integration Incompleteness***, ***Integration Redundancy***, and ***Integration Unsoundness***. For an integration step $\{C_1, C_2, ..., C_n\} \rightarrow C$, the first three error types originate from a intermediate conclusion C_i, including the presence of conclusions that contradicts prior findings,the absence of crucial conclusions, and the introduction of unnecessary or redundant conclusions. The final error type, namely Integration Unsoundness, refers to concluding a final conlusion C that is incorrect or unreasonable, even when integrated conclusions all satisfy soundness an completeness.

3.2 Benchmark Construction

The dataset construction pipeline comprises two core stages: **Socratic Reasoning Generation** and **Test Case Construction**.

3.2.1 Socratic Reasoning Generation

This stage aims to create a data pool of Socratic reasoning process, represented as a sequence of atomic Socratic reasoning actions. As illustrated in left part of Figure 2, each reasoning step is enclosed with a start tag <[Pattern]> and an end tag </[Pattern]>. The content within the [Pattern] placeholder indicates the specific reasoning pattern that characterizes this particular step.

Socratic Reasoning Model Training Given the scarcity of available Socratic reasoning data, we initially trained a specialized Socratic reasoning model to facilitate data generation. To achieve this, we sampled 19k instances from the MATH-Hard (Hendrycks et al., 2021) and Open-o1 (OpenO1, 2024) datasets and transformed their existing Chain-of-Thought (CoT) annotations into Socratic reasoning processes. We then fine-tune Qwen2.5-72b-instruct (Team, 2024a) on these Socratic reasoning processes, yielding our Socratic reasoning model, denoted as $M_{Socratic}$.

Socratic Reasoning Generation Subsequently, we leverage $M_{Socratic}$ to generate new Socratic reasoning processes from metadata. To this end, we first collect samples from GSM8k (Cobbe et al., 2021), Omni-Math (Gao et al., 2024), MathBench (Liu et al., 2024), and OlympiadBench (He et al., 2024a). In order to ensure that our problems are adequately challenging, we carefully curated the Omni-Math and MathBench datasets. Specifically, we excluded any Omni-Math samples with a difficulty rating lower than 4.0. For MathBench, we focused solely on MathBench-A, as this subset emphasizes theoretical application rather than conceptual understanding. Furthermore, we only retained instances from MathBench-A that are designated as high school or university level. This procedure finally results in a pool D. For each

	Overall	*Transformation*	*Decomposition*	*Regather*	*Deduction*	*Integration*	*Verification*
Avg. Steps	8.7	8.5	8.7	8.6	8.5	8.5	10.8
Avg. Error Steps	3.0	4.2	3.3	2.9	3.0	2.0	3.8
Avg. First Error Step	4.7	1.5	3.0	3.1	5.4	7.2	6.9
Avg. Question Length	209.6	224.4	220.7	207.5	221.7	191.3	169.4
# of Instances	2995	313	463	463	926	615	215

Table 2: Statistics of SOCRATIC-PRMBENCH.

question-answer pair (q_i, a_i) in D, $M_{Socratic}$ generates a Socratic reasoning process r_i, resulting in a $(q_i, r_i, \hat{a}_i)$ triplet.

Socratic Reasoning Curation Finally, each $(q_i, r_i, \hat{a}_i)$ tuple undergos a rigorous dual verification process: answer correctness was first assessed, followed by LLM-based verification of each individual step. Only tuples that pass both verifications are retained, resulting in our metadata set D'. For answer verification, we follow Qwen2.5-Math (Yang et al., 2024), requiring that the predicted answer $\hat{a}_i$ satisfies both numerical and symbolic equivalence with the ground truth answer a. For step verification, we leverage GPT-4o (OpenAI, 2024a) to assess the correctness of each individual step in the reasoning process, with the detailed prompt in Appendix B.

3.2.2 Test Case Constuction

In the stage, we generate test sets for each error type C (as classified in Section 3.1) by employing a controlled error injection procedure. For each error type C (e.g., Repeat Inconsistency), we create a test set T_C. This is achieved by first randomly select N samples from the metadata set D'. And then for each sample (q_i, r_i, a_i), including a problem q_i, a Socraitc reasoning path r guaranteed completely correct through our dual verification process, we prompt gpt-4o to modify the originally correct reasoning process r,intentionally introducing an error consistent with error type C:

$$\begin{aligned} \tilde{r}_j &= \text{LLM}(I, [q_j, r_j, a_j], C) \\ T_C &= \{t_j = (q_j, \tilde{r}_j, \tilde{a}_j)\}_{j=1}^{N} \end{aligned} \tag{1}$$

where $\tilde{r}_j$ is the modified socratic reasoning process with the type of error C and I is the instruction prompt for GPT-4o to modify original process r_j to $\tilde{r}_j$, with detailed prompt in Appendix B.

3.3 Quality Control

To ensure the high quality and reliability of SOCRATIC-PRMBENCH, we utilize both rule-based and LLM-based method to filter out any unsuitable samples, thereby ultimately creating our SOCRATIC-PRMBENCH.

Rule-based Fitering Despite providing detailed task descriptions and output format requirements in the instruction T, GPT-4o may still occasional fail to follow the instruction T strictly. Therefore, we implement a rule-based filtering method. First, we use string matching to identify and remove any sample that fails to produce output in JSON format, which is required in T. Second, we used regular expression to discard any sample that fail to successfully output the final answer.

LLM-based Filtering To ensure the quality of our generated test cases, we employ Gemini2.5-pro to evaluate each sample $(q_j, \tilde{r}_j, \tilde{a}_j)$, within a test set T_C for a given error type C. Specifically, we instruct Gemini2.5-pro to assess the sample based on two criteria: (1) that the reasoning path $\tilde{r}_j$ appears superficially plausible yet contains an underlying reasoning error, and (2) that the identified error should definitively belong to the targeted error type C, with detailed prompt shown in Appendix B. After filtering by Gemini2.5-pro, the acceptance rate of samples reached 92.7%, and 2995 samples were retained to form the final Socratic-PRMBench. The statistics of Socratic-PRMBench are shown in Table 2.

LLM's Consistency with Human Annotators To demonstrate Gemini2.5-pro's ability to perform this quality filtering task, we measure its agreement with human annotators. We recruit three volunteer annotators, each holding at least a bachelor's degree, and ask them to verify a randomly sampled 10% subset of our data using the exact same criteria with Gemini2.5-pro. We then calculate the agreement rate between Gemini2.5-pro and the human annotators. As a result, Gemini2.5-pro shows a high degree of consistency with the human annotators, achieving an average agreement rate of 93.3%. This high level of consistency provides strong evidence that Gemini2.5-pro can effectively replace human annotators in performing quality filtering

Model	Transformation		Decomposition			Regather			Verification	
	TT.	TF.	DC.	DR	DS.	GP.	GC.	GR.	CE.	DE.
Process Reward Models (PRMs)										
Skywork-PRM-7B	38.7	38.4	42.7	42.5	38.0	42.8	44.8	41.3	47.9	46.7
ReasonEval-7B	50.9	50.9	59.3	50.1	53.7	52.4	59.6	49.7	66.7	59.2
RLHFlow-PRM-Mistral-8B	50.6	52.7	46.6	47.3	42.7	38.0	44.6	48.7	53.1	49.5
RLHFlow-PRM-Deepseek-8B	47.5	50.8	50.6	50.9	44.0	41.6	48.6	55.4	45.9	47.6
MathShepherd-Mistral-7B	54.5	50.9	59.4	**57.4**	56.7	**60.9**	59.4	54.6	**72.7**	**72.1**
Qwen2.5-Math-PRM-7B	**55.8**	**64.3**	**61.7**	51.6	**58.4**	57.5	**61.8**	**58.2**	67.4	64.1
LLMs, Prompted as Critic Models										
GPT-4o	**62.4**	60.5	69.9	60.0	66.1	64.9	**74.1**	57.9	74.4	75.8
Deepseek-R1	51.9	**72.6**	63.4	64.4	67.1	70.9	64.6	54.8	75.0	77.1
QwQ-32B	60.2	68.6	70.0	**67.9**	59.8	73.7	65.8	55.4	75.8	75.7
Gemini-2.5-Pro	62.3	64.4	67.3	61.4	**68.5**	70.2	69.2	**58.6**	**78.3**	**78.0**
o3-mini	**62.4**	67.4	**70.4**	57.3	68.0	**77.3**	71.3	53.0	77.2	72.6

Model	Overall	Deduction						Integration			
		CF.	CT.	CV.	PC.	PR.	PS.	IC.	IT.	IR.	IS.
Process Reward Models (PRMs)											
Skywork-PRM-7B	43.6	42.5	41.2	40.0	41.8	42.8	39.8	38.7	42.6	39.4	44.2
ReasonEval-7B	61.9	63.6	63.6	66.3	61.9	65.2	63.5	69.7	78.2	68.7	76.1
RLHFlow-PRM-Mistral-8B	48.8	50.4	46.2	45.2	46.1	44.5	43.3	51.2	58.1	46.6	56.3
RLHFlow-PRM-Deepseek-8B	51.5	51.5	52.4	52.0	47.6	51.4	45.2	55.3	63.7	53.3	66.7
MathShepherd-Mistral-7B	64.4	68.0	65.9	66.5	62.4	65.9	65.4	63.1	74.2	60.1	72.3
Qwen2.5-Math-PRM-7B	**68.0**	**74.7**	**73.1**	**72.2**	**66.6**	**72.4**	**67.2**	**75.0**	**85.2**	**69.6**	**86.9**
LLMs, Prompted as Critic Models											
GPT-4o	70.8	63.6	62.7	74.5	73.2	60.1	76.1	73.4	80.8	52.7	88.7
Deepseek-R1	73.0	80.8	72.6	77.2	68.6	72.0	76.9	75.9	78.9	59.9	88.6
QwQ-32B	73.8	70.3	75.0	**85.2**	**74.0**	69.5	77.5	**81.8**	83.5	58.7	96.7
Gemini-2.5-Pro	73.5	72.8	77.7	83.5	69.0	65.9	73.5	73.2	**88.9**	56.9	**96.9**
o3-mini	**75.7**	**83.3**	**81.0**	81.4	73.9	**75.3**	**78.6**	78.7	87.3	**72.0**	87.0

Table 3: Evaluation results on SOCRATIC-PRMBENCH. (Up): The PRM-Score of *Transformation*, *Decomposition*, *Regather*, and *Verification*. (Down): The PRM-Score of *Deduction*, *Integration* and *Overall* performance. The best performance for each category and task is in **bold**. The full names of abbreviations are shown in Appendix A

across the entire dataset, reducing the burden of extensive manual work.

4 Experiments

4.1 Models

In our setting, we consider two types of model: Process Reward Models (PRMs) and Large Language Models (LLMs) prompted as critic models.

Process Reward Models (PRMs) are trained with annotations of intermediate reasoning steps to evaluate and supervise intermediate reasoning process of language models.

Our evaluation includes state-of-the-art open-source PRMs, such as: (1) MathShepherd (Wang et al., 2023), which obtains the process label for each step by estimating the empirical probability of that step leading to the correct final answer. (2) Two LLaMA-3.1-based Generative PRMs (Dong et al., 2024) that determine correctness based on the output probabilities of "Yes/No" tokens. (3) ReasonEval (Mondorf and Plank, 2024), which asses redundancy in addition to validity of reasoning steps. (4) Two PRMs trained on the popular mathematical model Qwen2.5-Math, namely Skywork-PRM (He et al., 2024b) and Qwen2.5-Math-PRM (Zhang et al., 2025).

Large Language Models (LLMs) Prompted as Critic Models Critic models aim to provide feedback and critique directly on model-generated texts, harnessing the generative power of Large Language Models. Our evaluation includes both general-purpose models, including GPT-4o (OpenAI, 2024a), Gemini2.5-Pro (Deepmind, 2025), and models specilized on reasoning, including Deepseek-R1 (DeepSeek-AI, 2025), QwQ-32B (Team, 2024b), and o3-mini (OpenAI, 2025).

4.2 Evaluation Metrics

Given that the evaluation of PRM centers on the detection of flawed reasoning steps, a straightforward application of Accuracy or F1-score may be affected by inherent biases of models. To address this concern, we follow (Song et al., 2025; Zheng et al., 2024) and employ the PRM-score as our

evlauation metric, defined formally as:

$$\text{PRM-Score} = w_1 \times \text{F1}_{\text{neg}} + w_2 \times \text{F1} \quad (2)$$

where F1 and F1_{neg} refer to F1 scores and negative F1 scores. w_1 and w_2 are weights that balance the contributions of the F1-score and negative F1-score. Following previous studies (Song et al., 2025; Zheng et al., 2024), we set $w_1 = w_2 = 0.5$.

4.3 Main Results

Our evluation results are exhibited in Table 3. Our findings are as follow:

Comparision between PRMs and LLMs The performance of PRMs is demonstrably inferior to that of LLMs. The top-performing PRM, Qwen2.5-Math-PRM-7B, achieves a score of only 68.0, which is lower than even the least effective LLM, gpt-4o. Furthermore, some PRMs perform below the level of random guess, highlighting their limitations in handling reasoning errors across diverse reasoning patterns. This suggests a considerable gap between PRMs and LLMs, indicating a need for substantial improvement. The challenges of PRM data annotation and the difficulty in ensuring the quality of synthetic data likely contribute to this disparity. For instance, Math-shepherd leverages synthetic data where step correctness is measured based on the estimated probability of arriving at the correct final answer, whereas Qwen2.5-Math-PRM-7B uses the manually labeled PRM800k dataset.

Comparision among LLMs In contrast to PRMs, LLMs exhibit the potential to provide more robust and reliable rewards in critique, owing to their sophisticated language and reasoning skills. Consistent with this, we observe that reasoning-specialized LLMs outperforms general-purpose LLMs. Notably, QWQ-32B performs best among the open-source models and even outperforms GPT-4o. While QWQ -32B demonstrates impressive performance, it still underperforms o3-mini, indicating that although the gap in problem-solving performance is getting closer between open-source and proprietary models, a significant gap persists in their capabilities as critic models.

Redundant errors are more challenging We observed notable performance variations across fine-grained error types, even within the same reasoning pattern. Redundant errors, such as decomposition redundancy, regather redundancy, and integration redundancy within the Decomposition, Regather, and Integration patterns, consistently posed a greater challenge for both PRMs and LLMs compared to other error types within the same reasoning pattern. This may be attributted that redundant error steps often appear more "normal" or plausible than other types of erroneous steps, hindering the models' ability to identify them based on surface-level textual cues. This suggests that current PRMs may be limited by their reliance on surface-level pattern recognition for error detection, highlighting the need for more profound reasoning and analytical capabilities.

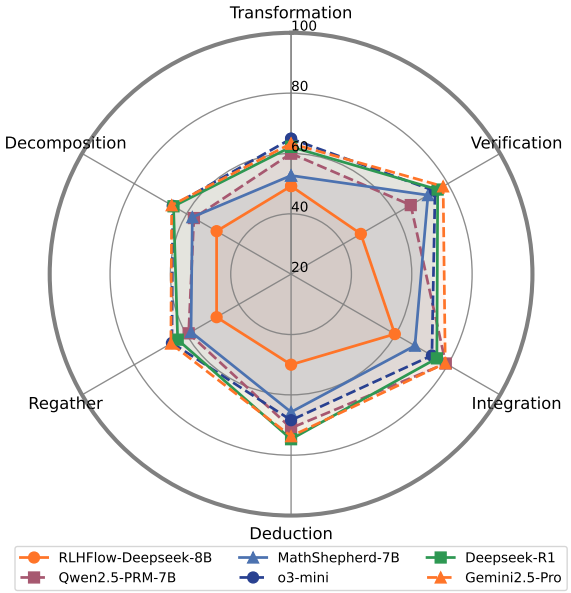

Figure 3: Average PRM-Score of representative PRMs and LLMs across 6 reasoning patterns. Both PRMs and LLMs shows imbalanced performance.

4.4 Detailed Analysis

This section delves into a more nuanced analysis of our proposed SOCRATIC-PRMBENCH, aiming to identify current models' limitations in providing process-level rewards and provide insights to guide the future development of PRMs.

Disparities in performance across reasoning patterns As shown in Figure 3, we present the average PRM-Scores of representative PRMs and LLMs across the six reasoning patterns. A notable finding is the imbalanced performance exhibited by both PRMs and LLMs across different reasoning patterns. The performance of almost all models was consistently weaker on *Transformation*, *Decomposition*, and *Regather* patterns compared to *Deduction*, *Integration*, and *Verification*. This issue is more pronounced for PRMs, for example,

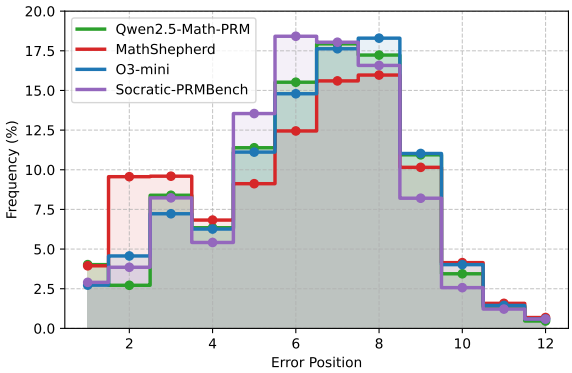

Figure 4: Error position distribution (truncated to 12) of SOCRATIC-PRMBENCH and the predicted error position distribution of several PRMs and LLMs.

Model	Accuracy			PRM Score
	Corr.	Err.	All.	
Random †	50.0	50.0	50.0	50.0
Process Reward Models (PRMs)				
ReasonEval-7B	87.3	35.7	69.6	61.9
Skywork-PRM-7B	22.7	93.0	44.5	43.6
MathShepherd	73.3	56.0	67.4	64.4
Qwen2.5-Math-PRM-7B	90.8	42.9	74.5	68.0
LLMs, Prompted as Critic Models				
GPT-4o	83.0	57.5	74.6	70.8
QwQ-32B	83.9	63.1	76.8	73.8
o3-mini	82.6	69.0	78.0	75.7
Gemini-2.5-Pro	83.6	62.8	76.5	73.5

Table 4: Comparison of model performance on positive and negative test cases.† represents performance of Random Guess.

Qwen2.5-Math-PRM-7B achieved a PRM-Score close to 80.0 on the *Integration* pattern but struggles to reach 60.0 on the *Decomposition* pattern. This finding highlights a potential bias in the current PRM training data construction process. Existing PRM datasets, regardless of whether they're manually annotated or synthetically generated, appear to lack adequate representation of different reasoning patterns. Due to the greater frequency of certain patterns like *Deduction*, these datasets tend to be dominated by those patterns, resulting in significantly worse performance on rarer patterns such as *Decomposition*. This observation underscores the importance of considering the distribution of different reasoning patterns in future PRM training data construction, as early detection of reasoning errors is critical to mitigate error propagation.

Models show latency in indentifying error steps To investigate the ability of models to detect reasoning errors in time, we compared the distribution of the ground truth error step positions in SOCRATIC-PRMBENCH with the distributions of predicted error positions for representative PRMs and LLMs. As evidenced by Figure 4, Qwen2.5-Math-PRM and o3-mini show a marked shift towards later steps compared to the ground truth distribution, indicating a delay in detecting early errors. This implies a limited ability to detect errors early on, allowing them to propagate. On the other hand, MathShepherd exhibits an opposite trend, with its predicted distribution shifts toward the beginning of the reasoning chain, suggesting that MathShepherd is prone to falsely identifying correct steps as errors, especially in the early stages of reasoning. This inspires us that both early detection and avoidance of excessive false positives are crucial. Although propagation of errors will waste computational resources and reduces sampling efficiencys, overly aggressive error detection can prematurely terminate correct reasoning paths, hindering the exploration of potentially optimal solutions.

Reward Bias of PRMs Table 3 reveals that some PRMs perform even worse than random guessing, suggesting a substantial bias in their predictions. To further quantify this bias, we calculated accuracy for correct and error reasoning steps for each model. As shown in Table 4, the results reveal a clear reward bias within PRMs, with some models heavily favoring positive rewards and others tending to provide negative rewards. For instance, Qwen2.5-Math-PRM-7B displays a 90.8% accuracy on correct steps but only a 42.9% accuracy on error steps. In stark contrast, Skywork-PRM-7B shows a 93.0% accuracy on error steps but only a 22.7% accuracy on correct steps. While LLMs exhibits less pronounced bias than PRMs, however, a considerable gap remained in accuracy between correct and error steps. Moreover, all the evaluated LLMs tended to favor positive rewards, which may limit their reliability in identifying subtle errors when serve as critic models.

5 Conclusion

In this work, we propose SOCRATIC-PRMBENCH, a systematic and fine-grained benchmark for PRMs. SOCRATIC-PRMBENCH comprises 2995 instances, categorized into six primary reasoning patterns and 20 sub-categories of fine-grained error types. Through a systematic and comprehensive evaluations of existing PRMs and LLMs prompted as critic models, we observe potential shortcomings in existing models and provide valuable insights for future efforts on upgrading PRMs.

Limitations

Although our work can provide a systematic and comprehensive evaluationg for PRMs, the current version of our benchmark primarily focuses on reasoning tasks with objectively verifiable answers, such as mathematical problem. Applying our existing data construction methods to tasks in domains like literature, medicine, or law, where definitive ground truth is often absent, needs further exploration. We intend to expand our benchmark to encompass a broader range of tasks in future versions of our benchmark.

References

Zachary Ankner, Mansheej Paul, Brandon Cui, Jonathan D. Chang, and Prithviraj Ammanabrolu. 2024. Critique-out-loud reward models. *Preprint*, arXiv:2408.11791.

Hritik Bansal, Arian Hosseini, Rishabh Agarwal, Vinh Q. Tran, and Mehran Kazemi. 2025. Smaller, weaker, yet better: Training LLM reasoners via compute-optimal sampling. In *The Thirteenth International Conference on Learning Representations*.

Sanjiban Choudhury. 2025. Process reward models for llm agents: Practical framework and directions. *arXiv preprint arXiv:2502.10325*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Deepmind. 2025. Gemini2.5-pro. https://deepmind.google/technologies/gemini/pro/.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Hanze Dong, Wei Xiong, Bo Pang, Haoxiang Wang, Han Zhao, Yingbo Zhou, Nan Jiang, Doyen Sahoo, Caiming Xiong, and Tong Zhang. 2024. RLHF workflow: From reward modeling to online RLHF. *Transactions on Machine Learning Research*.

Qingxiu Dong, Li Dong, Ke Xu, Guangyan Zhou, Yaru Hao, Zhifang Sui, and Furu Wei. 2023. Large language model for science: a study on p vs. np. *arXiv preprint arXiv:2309.05689*.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Runxin Xu, Zhengyang Tang, Benyou Wang, Daoguang Zan, Shanghaoran Quan, Ge Zhang, Lei Sha, Yichang Zhang, Xuancheng Ren, Tianyu Liu, and Baobao Chang. 2024. Omni-math: A universal olympiad level mathematic benchmark for large language models. *Preprint*, arXiv:2410.07985.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. 2024a. OlympiadBench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3828–3850, Bangkok, Thailand. Association for Computational Linguistics.

Jujie He, Tianwen Wei, Rui Yan, Jiacai Liu, Chaojie Wang, Yimeng Gan, Shiwen Tu, Chris Yuhao Liu, Liang Zeng, Xiaokun Wang, Boyang Wang, Yongcong Li, Fuxiang Zhang, Jiacheng Xu, Bo An, Yang Liu, and Yahui Zhou. 2024b. Skywork-o1 open series. https://huggingface.co/Skywork.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *Preprint*, arXiv:2106.09685.

Yixin Ji, Juntao Li, Hai Ye, Kaixin Wu, Jia Xu, Linjian Mo, and Min Zhang. 2025. Test-time computing: from system-1 thinking to system-2 thinking. *arXiv preprint arXiv:2501.02497*.

Nathan Lambert, Valentina Pyatkin, Jacob Morrison, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, and 1 others. 2024. Rewardbench: Evaluating reward models for language modeling. *arXiv preprint arXiv:2403.13787*.

Changcheng Li, Xiangyu Wang, Qiuju Chen, Xiren Zhou, and Huanhuan Chen. 2024. Mtmt: Consolidating multiple thinking modes to form a thought tree for strengthening llm. *arXiv preprint arXiv:2412.03987*.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*.

Zicheng Lin, Zhibin Gou, Tian Liang, Ruilin Luo, Haowei Liu, and Yujiu Yang. 2024. CriticBench: Benchmarking LLMs for critique-correct reasoning. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 1552–1587, Bangkok, Thailand. Association for Computational Linguistics.

Hongwei Liu, Zilong Zheng, Yuxuan Qiao, Haodong Duan, Zhiwei Fei, Fengzhe Zhou, Wenwei Zhang,

Songyang Zhang, Dahua Lin, and Kai Chen. 2024. MathBench: Evaluating the theory and application proficiency of LLMs with a hierarchical mathematics benchmark. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 6884–6915, Bangkok, Thailand. Association for Computational Linguistics.

Yantao Liu, Zijun Yao, Rui Min, Yixin Cao, Lei Hou, and Juanzi Li. 2025. RM-bench: Benchmarking reward models of language models with subtlety and style. In *The Thirteenth International Conference on Learning Representations*.

Yingwei Ma, Yongbin Li, Yihong Dong, Xue Jiang, Rongyu Cao, Jue Chen, Fei Huang, and Binhua Li. 2025. Thinking longer, not larger: Enhancing software engineering agents via scaling test-time compute. *arXiv preprint arXiv:2503.23803*.

Philipp Mondorf and Barbara Plank. 2024. Beyond accuracy: Evaluating the reasoning behavior of large language models - a survey. In *First Conference on Language Modeling*.

OpenAI. 2024a. Gpt-4o system card. https://cdn.openai.com/gpt-4o-system-card.pdf. Accessed: 2024-09-26.

OpenAI. 2024b. Learning to reason with llms. https://openai.com/index/learning-to-reason-with-llms/.

OpenAI. 2025. Openai o3-mini system card. https://openai.com/index/o3-mini-system-card/.

OpenO1. 2024. Open-o1. https://opensource-o1.github.io/.

Jingyuan Qi, Zhiyang Xu, Ying Shen, Minqian Liu, Di Jin, Qifan Wang, and Lifu Huang. 2023. The art of socratic questioning: Recursive thinking with large language models. *Preprint*, arXiv:2305.14999.

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

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2025. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*.

Mingyang Song, Zhaochen Su, Xiaoye Qu, Jiawei Zhou, and Yu Cheng. 2025. Prmbench: A fine-grained and challenging benchmark for process-level reward models. *arXiv preprint arXiv:2501.03124*.

Qwen Team. 2024a. Qwen2.5: A party of foundation models.

Qwen Team. 2024b. Qwq: Reflect deeply on the boundaries of the unknown.

Guiyao Tie, Zeli Zhao, Dingjie Song, Fuyang Wei, Rong Zhou, Yurou Dai, Wen Yin, Zhejian Yang, Jiangyue Yan, Yao Su, and 1 others. 2025. A survey on post-training of large language models. *arXiv preprint arXiv:2503.06072*.

Luong Trung, Xinbo Zhang, Zhanming Jie, Peng Sun, Xiaoran Jin, and Hang Li. 2024. ReFT: Reasoning with reinforced fine-tuning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7601–7614, Bangkok, Thailand. Association for Computational Linguistics.

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, Bangkok, Thailand. Association for Computational Linguistics.

Peiyi Wang, Lei Li, Zhihong Shao, RX Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2023. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. *arXiv preprint arXiv:2312.08935*.

Guangzhi Xiong, Qiao Jin, Xiao Wang, Yin Fang, Haolin Liu, Yifan Yang, Fangyuan Chen, Zhixing Song, Dengyu Wang, Minjia Zhang, and 1 others. 2025. Rag-gym: Optimizing reasoning and search agents with process supervision. *arXiv preprint arXiv:2502.13957*.

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*.

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. 2024. Generative verifiers: Reward modeling as next-token prediction. In *The 4th Workshop on Mathematical Reasoning and AI at NeurIPS'24*.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2025. The lessons of developing process reward models in mathematical reasoning. *arXiv preprint arXiv:2501.07301*.

Jian Zhao, Runze Liu, Kaiyan Zhang, Zhimu Zhou, Junqi Gao, Dong Li, Jiafei Lyu, Zhouyi Qian, Biqing Qi, Xiu Li, and Bowen Zhou. 2025. Genprm: Scaling test-time compute of process reward models via generative reasoning. *Preprint*, arXiv:2504.00891.

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

Zihao Zhou, Shudong Liu, Maizhen Ning, Wei Liu, Jindong Wang, Derek F. Wong, Xiaowei Huang, Qiufeng Wang, and Kaizhu Huang. 2025. Is your model really a good math reasoner? evaluating mathematical reasoning with checklist. In *The Thirteenth International Conference on Learning Representations*.

A Experimental Details

Abbreviation of Sub-Categories The full names of abbreviations used in our experiments are shown in Table 5.

Abbr.	Full Name	Reasoning Pattern
TT.	Transformation Inconsistency	Tansformation
TF.	Transformation Counter-Factuality	Transformation
DC.	Decomposition Incompleteness	Decomposition
DR.	Decomposition Redundancy	Decomposition
DS.	Decomposition Unsoundness	Decomposition
GP.	Regather Imprecision	Regather
GC.	Regather Incompleteness	Regather
GR.	Regather Redundancy	Regather
CE.	Correction Error	Verification
DE.	Detection Error	Verification
CF.	Conclusion Counter-Factuality	Deduction
CT.	Conclusion Inconsistency	Deduction
CV.	Conclusion Invalidity	Deduction
PC.	Premise Incompleteness	Deduction
PR.	Premise Redundancy	Deduction
PS.	Premise Unsoundness	Deduction
IC.	Integration Incompleteness	Integration
IT.	Integration Inconsistency	Integration
IR.	Integration Redundancy	Integration
IS.	Integration Unsoundness	Integration

Table 5: The abbreviation of sub-ategorie impact of ICL few-shot numbers on models' final performance. The number reported here is PRMScore.

Implementation Details For Socratic reasoning model, we use LoRA tuning (Hu et al., 2021) to fine-tune a Qwen2.5-72B-Instruct with LLaMA-Factory library[1]. For the evaluation of open-source PRMs, we utilize PRM Eval ToolKit[2] for implementation. For the evalutation of LLMs prompted as critic models, we prompt LLMs with the prompt template in Table 6. During the test case construction procedure, we select $N = 150$ samples from metadata set D', including 10 from GSM8k and 50 each from Omni-Math, MathBench, and OlympiadBench. We release our dataset in https://anonymous.4open.science/r/Socratic-PRMBench-B8EF.

B Prompts

As described in Section 3, LLMs play a crutial role in our method. In the socratic reasoning generation stage, the prompt for socratic reasoning curation is illustrated in Table 7. In the test case construction stage, we follow (Song et al., 2025) and design task prompt and output format prompt seperately, as shown in Table 8 and Table 9 respectively. For the LLM-based filering procedure, we use the prompt template in Table 10.

[1]https://github.com/hiyouga/LLaMA-Factory
[2]https://github.com/ssmisya/PRMBench

Prompt Template for Evaluation of LLMs prompted as critic models

[System Prompt]
You are a mathematical reasoning evaluator. Your task is to analyze mathematical problem-solving steps and provide structured assessments in JSON format.

For each solution step, you need to evaluate its Validity Score (-1 to +1):
* +1: Completely correct mathematical reasoning
* 0: Partially correct with some mistakes
* -1: Completely incorrect
* Use any value in between to indicate varying degrees of correctness

Requirements:
- Evaluate each step independently
- Provide scores as floating-point numbers
- Return results in strict JSON format: {"validity ": [scores]}
- Ensure the array have the same length as the number of steps
- Maintain mathematical rigor in your evaluation
- Consider mathematical accuracy, logical coherence, and solution efficiency

Example output format:

{"validity ": [0.8, -0.5, 1.0]}

You will be presented with a mathematical problem and its step-by-step solution. Please analyze each step and provide your evaluation in the specified JSON format.

[User]
Question: {question}

Solutions: {solution}

Table 6: Prompt template for evaluation of LLMs prompted as critic models

Prompt Template for Step Verification

You are an expert on reasoning process verification, you will be given a question, a solution(split into paragraphs, enclosed with tags and indexed from 1, and a reference answer.

[Question]
{question}

[Solution]
{solution}

[Reference Answer]
{answer}

Your task is to review and critique the solution paragraph by paragraph. Once you identify an error in a paragraph, return the index of the paragraph where the earliest error occurs. Otherwise, return the index of -1 (which typically denotes "not found"). Please put your final answer (i.e., the index) in \boxed{}.

Table 7: Prompt template for step verification.

Task Prompt for Test Case Construction

You are a helpful AI assistant that is very good at reasoning and data construction. Now I want to test the ability of process-level reward models to judge whether a step within reasoning process is correct. To do this, please help me build flawed cases by introducing specific types of errors into a given reasoning process.
You will be provided with:
1. A mathematical problem.
2. A correct step-by-step reasoning process used to solve it. Each step is in a form of Action, posssibly including [Transformation], [Decomposition], [Regather], [Deduction], [Verification], [Integration], [Answer], [LVerification] and [GVerification].

The description of Actions are as follows:

[Transformation] (Identifier: <Repeat>xxx</Repeat>)
- Explain the problem from a problem-solving perspective
- Gain a more comprehensive and clear understanding of the problem through rephrasing

[Decomposition] (Identifier: <Decomposition>xxx</Decomposition>)
- Break down the problem into several core sub-problems; resolve the main problem by tackling each sub-problem
- If no breakdown is necessary, provide the solution approach

[Regather] (Identifier: <Regather>xxx</Regather>)
- Collect key information from the input relevant to solving the problem
- Output definitions, principles, and other concepts related to solving the problem, and provide explanations

[Deduction] (Identifier: <Deduction>xxx</Deduction>)
- Observe existing information and extract key parts
- Identify explicit and implicit requirements, considering constraints and limitations
- Propose concrete ideas for solving the problem
- Execute reasoning according to the ideas

[Verification]&[Verification] (Identifier: <Verification>xxx</Verification>)
- Verify the logical consistency of the reasoning process
- Check the reasoning process against existing evidence
- Look for potential flaws in the reasoning process and refine them
- Review the completeness of understanding
- Question your assumptions and consider alternative viewpoints

[Integration] (Identifier: <Integration>xxx</Integration>)
- Integrate all current reasoning processes to form the current conclusion

[Answer] (Identifier: <Answer>xxx</Answer>)
- Output the final answer to the original problem

Your task is to modify the question, adjust original steps, or introduce additional steps into the original process chain to create a reasoning process that appears plausible but is incorrect, which leads to a wrong answer. The objective is to simulate flawed solutions by incorporating the specified error detailed after '### Error Type to Introduce'.

Error Type to Introduce
{Error type}

Table 8: Task prompt for test case construction.

Output Foramt Prompt for Test Case Construction

Formatting Instructions:

After making the modifications, provide the following structured output:
{
"original_question": "The original mathematical problem.",
"modified_question ": "The modified problem or original problem, "
"original_process": ["original_step 1 ", "original_step 2", . . .],
"modified_process": ["modified_step 1", "modified_step 2 ", . . .],
"modified_steps": [1, 5, 7, . . .],
"error_steps": [5, 6, . . .],
"reason": "Explanation for the changes."
}

Detailed Requirements:
1. original_question: A string representing the original mathematical problem as provided.
2. modified_question: A string representing the modified problem after your changes. If the problem remains the same, you can copy the original question.
3. original_process: A non-empty list of strings representing the original reasoning steps provided as input.
4. modified_process: A non-empty list of strings representing the reasoning process after your modifications.
5. modified_steps: A non-empty list of integers indicating the indexes of all modified steps. Indexing starts at 1.
6. error_steps: A non-empty list of integers representing the steps that contain hallucinations or errors. These should also be part of modified_steps.
7. reason: A clear explanation of the modifications made, why they were introduced, and how they align with the specified error types.

Notes:
1. Ensure all lists are non-empty.
2. Use LaTeX format for all mathematical symbols (e.g., x^2 for x squared). Do not use Unicode symbols such as \u2248 or \u00f7.
3. Ensure the JSON object is well-formed, with proper escaping for special characters like backslash n (e.g., use backslash backslash n for newlines).
4. All indexes start from 1, that is, the first step's index is 1, not 0.
5. You can choose to modify the question or not, if the question remains the same, you can copy the original question. But if the question is modified, ensure that the steps is judged based on the modified question.
6. Please give original process as provided by the prompt, do not modify it.

Table 9: Output prompt for test case construction.

Prompt Template for LLM-based Filtering

You are an expert on reasoning process verification, you will be given a question, a solution(split into paragraphs, enclosed with tags.

Your task is to decide whether the step-by-step solution generated by LLMs satisfies:
1. The process generated by LLMs seems like a possible solution path that could happen.
2. The process generated by LLMs is exactly wrong and the type of error is suitable for the description of [classification]

[Classification]
{classification}

[Question]
{question}

[Solution]
{Solution}

Please answer a "Yes" if both of the two aspects are satisfied, otherwise answer 'No'.
Please put your final answer (Yes or No) in \boxed {}.

Table 10: Prompt template for LLM-based Filtering