PROCESSBENCH: Identifying Process Errors in Mathematical Reasoning

Chujie Zheng Zhenru Zhang Beichen Zhang Runji Lin Keming Lu
Bowen Yu* Dayiheng Liu* Jingren Zhou Junyang Lin*
Qwen Team, Alibaba Inc.

https://github.com/QwenLM/ProcessBench

Abstract

As language models regularly make mistakes when solving math problems, automated identification of errors in the reasoning process becomes increasingly significant for their scalable oversight. In this paper, we introduce **PROCESSBENCH** for measuring the ability to identify erroneous steps in mathematical reasoning. It consists of 3,400 test cases, primarily focused on competition- and Olympiad-level math problems. Each test case contains a step-by-step solution with error location annotated by human experts. Models are required to identify the earliest step that contains an error, or conclude that all steps are correct. We conduct extensive evaluation on PROCESSBENCH, involving two types of models: *process reward models (PRMs)* and *critic models*, where for the latter we prompt general language models to critique each solution step by step. We draw two main observations: (1) Existing PRMs typically fail to generalize to more challenging math problems beyond GSM8K and MATH. They underperform both critic models (i.e., prompted general language models) and our own trained PRM that is straightforwardly fine-tuned on the PRM800K dataset. (2) The best open-source model, QwQ-32B-Preview, has demonstrated the critique capability competitive with the proprietary model GPT-4o, despite that it still lags behind the reasoning-specialized o1-mini. We hope PROCESSBENCH can foster future research in reasoning process assessment, paving the way toward scalable oversight of language models.

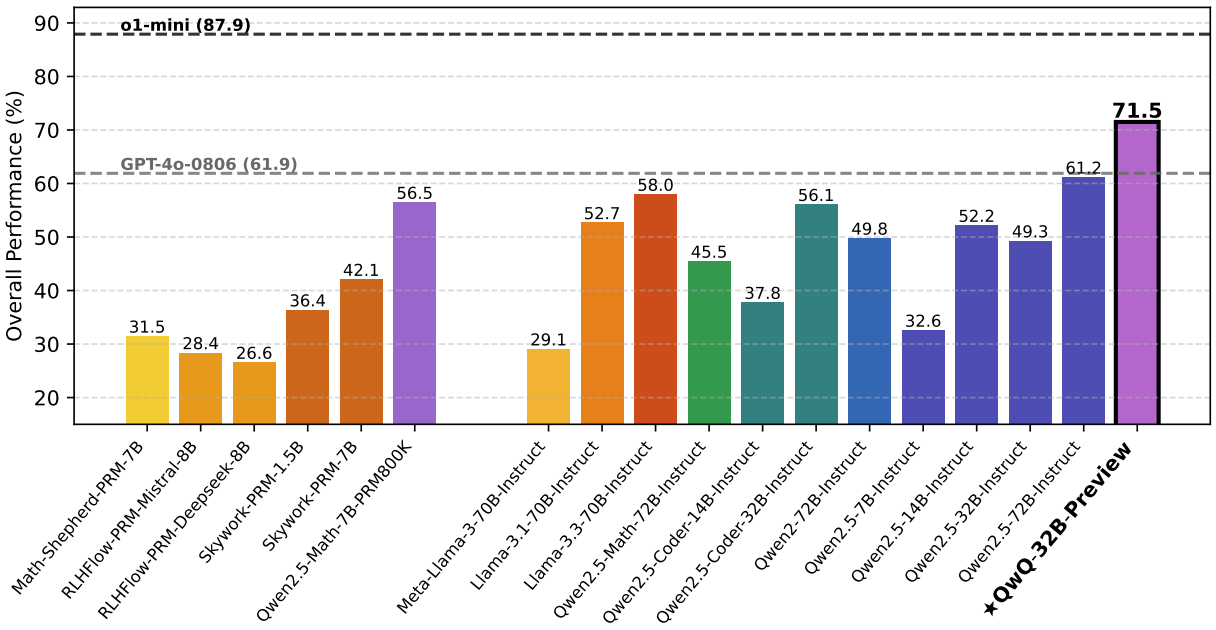

Figure 1: Overview of evaluation results on PROCESSBENCH (see Table 3 for details).

*Corresponding authors.

1 Introduction

In recent years, language models have made remarkable progress in complex reasoning tasks, such as mathematics and programming (Hurst et al., 2024; OpenAI, 2024; Yang et al., 2024a;b; Dubey et al., 2024; Wake et al., 2024), yet they still make mistakes when solving challenging problems. To achieve scalable oversight (Amodei et al., 2016; Bowman et al., 2022; Cao et al., 2024), i.e., effectively supervising AI systems that get close to or go beyond broadly human-level performance, particularly in complex tasks that are difficult for general humans, we expect language models can identify errors in their reasoning process in an automated way. However, existing benchmarks related to assessing language models' reasoning process may be hard to satisfy the growing evaluation demand for the error identification ability. Either their covered problems have become less challenging for recent language models (Zhou et al., 2024; Lightman et al., 2023), or they merely label the correctness of final answers but lack annotations for specific erroneous steps (Lin et al., 2024).


```
"id": "math-22",
"generator": "Qwen2-7B-Instruct",
"problem": "Find the integer $n,$ $0 \\le n \\le 180,$ such that $\\cos n^\\circ = \\cos
568^\\circ.$",
"steps": [
    "To find the integer \\(n\\), \\(0 \\leq n \\leq 180\\), such that \\(\\cos n^\\circ = \\cos
    568^\\circ\\), we can use the periodic properties of the cosine function. First, recall that
    the cosine function has a period of \\(360^\\circ\\). This means that: \\[ \\cos \\theta =
    \\cos (\\theta + 360k) \\] for any integer \\(k\\).",
    "Given \\(\\cos 568^\\circ\\), we need to reduce this angle to an equivalent angle within the
    first cycle of \\(0\\) to \\(360^\\circ\\). We do this by subtracting multiples of
    \\(360^\\circ\\) until we get an angle within this range: \\[ 568^\\circ - 360^\\circ =
    208^\\circ \\] So, \\(\\cos 568^\\circ = \\cos 208^\\circ\\).",
    "However, we want to find \\(n\\) such that \\(0 \\leq n \\leq 180\\). Since cosine is also
    symmetric about \\(180^\\circ\\), we know that: \\[ \\cos \\theta = \\cos (180^\\circ -
    \\theta) \\] Therefore, \\(\\cos 208^\\circ = \\cos (180^\\circ - 208^\\circ)\\), which
    simplifies to: \\[ \\cos 208^\\circ = \\cos (-28^\\circ) \\]",
    "The cosine function is also even, meaning it is symmetric about the y-axis: \\[ \\cos (-
    \\theta) = \\cos \\theta \\] Thus, \\[ \\cos (-28^\\circ) = \\cos 28^\\circ \\]",
    "So, \\(n = 28^\\circ\\). Hence, the integer \\(n\\), \\(0 \\leq n \\leq 180\\), such that
    \\(\\cos n^\\circ = \\cos 568^\\circ\\) is \\(n = 28\\). The answer is \\(\\boxed{28}\\)."
],
"final_answer_correct": false,
"label": 2
```


Figure 2: Data example of PROCESSBENCH. The label 2 denotes that the earliest error occurs in the **2nd step** (indexed from 0). For test cases with no errors, the labels are -1.

In this paper, we introduce **PROCESSBENCH** for measuring the ability to identify erroneous steps in mathematical reasoning. Figure 2 presents a data example. We prioritize several principles when designing this benchmark:

- **Problem difficulty and solution diversity.** PROCESSBENCH primarily covers competition- and Olympiad-level math problems and utilizes various open-source language models to generate solutions. This ensures both the difficulty of math problems and the diversity of solution styles, enabling robust evaluation.
- **Scale and accuracy.** PROCESSBENCH consists of 3,400 test cases, with all solutions annotated with error locations by multiple human experts. The large scale and expert annotation ensure the data quality and the reliability of evaluation.
- **Simplicity.** PROCESSBENCH requires models to identify the earliest erroneous step occuring in the solution, if any exists. This straightforward evaluation protocol enables easy adaptation for various types of models, such as process reward models (PRMs) and critic models.

We conduct extensive evaluation on PROCESSBENCH, involving two types of models: *process reward models (PRMs)* and *critic models*. For PRMs, we include multiple open-source PRMs (Wang et al., 2024; Skywork, 2024; Xiong et al., 2024b) to assess the correctness of each reasoning step in the solution. For critic models, we prompt general language models like Qwen (Yang et al., 2024a; Qwen, 2024a; Hui et al., 2024) and GPT-4o (Hurst et al., 2024) to critique each solution step by step. We show that, despite recent growing interest, existing PRMs typically fail to generalize to more challenging math problems

beyond GSM8K and MATH. They underperform both critic models and our own trained PRM that is straightforwardly fine-tuned on the PRM800K dataset, which raises questions about the generalization abilities and scalability of the current data synthesis methodologies used to build PRMs. In contrast, general language models manifest non-trivial critique capabilities that can not only identify erroneous steps but also provide detailed explanations. The best open-source model, QwQ-32B-Preview (Qwen, 2024b), has performed competitively with the proprietary GPT-4o model, while it still lags behind the reasoning-specialized o1-mini (OpenAI, 2024). We hope PROCESSBENCH can catalyze future research in automated reasoning process assessment, establishing crucial foundations for scalable oversight of language models.

2 Related Work

There exist several benchmarks or datasets related to assessing language models' reasoning process. CriticBench (Lin et al., 2024) evaluates language models' abilities to critique solutions and correct mistakes in various reasoning tasks. MathCheck (Zhou et al., 2024) synthesizes solutions containing erroneous steps using the GSM8K dataset (Cobbe et al., 2021), in which language models are tasked with judging the correctness of final answers or reasoning steps. PRM800K (Lightman et al., 2023) builds on the MATH problems (Hendrycks et al., 2021) and annotates the correctness and soundness of reasoning steps in model-generated solutions. It also has sparked a blooming of research interest in building process reward models (PRMs) (Wang et al., 2024; Xiong et al., 2024b;a).

Table 1: Comparison between PROCESSBENCH and other benchmarks or datasets related to reasoning process assessment (Lin et al., 2024; Zhou et al., 2024; Lightman et al., 2023). †: Solution diversity denotes the diversity of language models used for solution generation, corresponding to the "# Solution Generators" column. ‡: For PRM800K, we only count the 90 complete solutions in its phase 1 test set, as the complete solutions in its phase 2 test set are all terminated at the earliest erroneous steps.

	Problem Diffculty	# Solution Generators	Solution Diversity†	Step Annotation?	Annotator	Test Case Size (Identifying Process Errors)
CriticBench	★★	8	★★★	✗	-	-
MathCheck-GSM	★	1	★	✓	Synthetic	516
PRM800K	★★	1	★	✓	Human	90‡
PROCESSBENCH	★★★	12	★★★	✓	Human	3,400

PROCESSBENCH is distinguished from prior benchmarks or datasets in three key aspects, as highlighted in Table 1. **First**, PROCESSBENCH primarily covers more challenging math problems with competition- or Olympiad-level difficulty, which better fit the rapidly growing capabilities of modern language models. **Second**, rather than relying on synthetic data, PROCESSBENCH leverages diverse model-generated natural solutions and employs expert annotation to label erroneous steps, which ensures both real-world applicability and label accuracy. **Third**, the large scale of PROCESSBENCH (3,400 test cases in total) enables more comprehensive and robust evaluation.

There has also been extensive research on language models' scalable oversight (Amodei et al., 2016; Bowman et al., 2022; Cao et al., 2024) and studies on whether language models can identify the errors in their own outputs. Lightman et al. (2023); Wang et al. (2024); Luo et al. (2024) propose to train specialized reward models to supervise language models' reasoning process (i.e., process reward models or PRMs). Huang et al. (2023); Kamoi et al. (2024) argue that general language models struggle to identify and correct their reasoning errors without external feedback. Saunders et al. (2022); McAleese et al. (2024) show that language models can be trained to write informative critiques for both assisting human evaluation and enabling self-refinement, which favorably scales with increased model capabilities (or model sizes). We believe the improved capabilities of error identification will build strong foundations for language models' scalable oversight.

3 Benchmark Construction

3.1 Task Definition

As shown in Figure 2, given a math problem and a step-by-step solution, PROCESSBENCH requires models to either identify the *earliest-occurring error*, or conclude that all steps are *correct*. Formally, given a math problem P and its step-by-step solution $S = \{s_0, ..., s_{n-1}\}$, the task is to output an index

$i \in \{-1, 0, ..., n-1\}$, where $i = -1$ indicates that all steps are correct, and $i \geq 0$ indicates that the earliest error occurs at step s_i.

Typically but non-inclusively, we consider a step as erroneous if it contains any of the following: (1) **Mathematical errors**: incorrect calculations, algebraic manipulations, or formula applications. (2) **Logical errors**: invalid deductions, unwarranted assumptions, or flawed reasoning steps. (3) **Conceptual errors**: misunderstanding or misapplication of mathematical or problem concepts. (4) **Completeness errors**: missing crucial conditions, constraints, or necessary justifications that affect the solution's validity. Beyond these types of errors, we encourage human annotators to determine the correctness of reasoning steps based on their own expertise. We do not require human annotators to explicitly annotate error types due to the intractability of intentional categorization.

Note that for steps after the first error, the meaning of their correctness may become ambiguous or debatable: derivations based on incorrect premises can make sense, but still remain on a globally incorrect reasoning path (Lightman et al., 2023). For instance, if step k contains an error in calculating $x = 2$, when it should be $x = 3$, subsequent steps may follow valid algebraic rules but operate on an incorrect value of x, making their individual correctness hard to determine. This is why PROCESSBENCH focuses on identifying the earliest-occurring error in the reasoning process.

3.2 Benchmark Construction

Problem Curation We collect math problems from the test sets of four public and widely used datasets in mathematical reasoning tasks: GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021), OlympiadBench (He et al., 2024), and Omni-MATH (Gao et al., 2024). Except for GSM8K, which consists of grade school math problems, the other three datasets all contain problems with competition- or Olympiad-level difficulty.

Solution Generation We generate solutions using the widely used Qwen (Yang et al., 2024a; Qwen, 2024a; Yang et al., 2024b) and LLaMA (Dubey et al., 2024) series open-source models, resulting in twelve distinct solution generators in total. This includes a wide range of model families, sizes, and downstream task performance, leading to the high diversity of solution styles. Table 4 in Appendix B presents the breakdown of language models used for PROCESSBENCH's solution generation.

Solution Reformatting In mathematical reasoning tasks, double line breaks (i.e., "\n\n") are commonly used to segment solution steps (or paragraphs). However, we observed inconsistent step granularity due to varying solution styles and generation randomness. Some generated solutions frequently used double line breaks, resulting in numerous short, logically incomplete steps, while others used them sparingly, leading to lengthy paragraphs that combine multiple logical components. Such inconsistency in step granularity (and potential improper step segmentation) would impede the standardization of human annotation criteria.

To address this issue, we adopt a *solution reformatting* method to standardize the step granularity, through which the segmented paragraphs can better correspond to logically complete and progressive reasoning steps. Specifically, we first replace all the line breaks with white space, and then ask Qwen2.5-72B-Instruct to insert double line breaks (i.e., segment paragraphs) while preserving the solution content. Since we found that Qwen2.5-72B-Instruct sometimes alters the solution content ($< 0.5\%$), we remove those solutions whose final answers change after reformatting (although the content alteration may not influence benchmark construction). Consequently, the reformatting method effectively unifies the step granularity. Figure 6 in Appendix A presents an example of solution reformatting.

Expert Annotation To ensure a balance between erroneous and correct solutions, we first use Qwen2.5-72B-Instruct to verify the correctness of final answers in the model-generated solutions against the reference answers. We then respectively sample solutions with correct or incorrect final answers for subsequent annotation in a balanced way to avoid excessive concentration on solutions from either the weakest or strongest models.

We recruit human experts with doctoral-level mathematical expertise for annotation, and all of them are required to pass the mandatory proficiency examination and annotation tutorial. The annotators are designated with the same task in § 3.1, i.e., identifying the earliest-occurring error in each solution. However, we notice that the competition- or Olympiad-level math problems can still be challenging even for doctoral students majoring in mathematics. According to the feedback from the annotators, although they were not required to solve problems from scratch but rather to identify erroneous steps in presented solutions, they would still become quite hesitant in their annotations when uncertain about the correct solution approach, which affected both the annotation speed and quality. To ease the annotation difficulty,

we provide annotators with the reference solutions and answers from the original datasets, while we still explicitly instructed them to inspect and verify the presented model-generated solutions step by step.

Each solution is initially assigned to three different experts. When the initial three annotators cannot reach consensus, we increase the number of annotators until three of them agree on the same result. If an agreement cannot be achieved within five annotators (e.g., annotation distribution of $(2, 2, 1)$ or $(2, 1, 1, 1)$), we discard this solution. This leads to an overall $\sim 30\%$ discard rate throughout the entire annotation process. We also discard the solutions where the final answers are incorrect (according to the reference answers) but the human annotation results are correct. Although such cases are fairly rare ($< 1\%$), they are mostly concentrated in the OlympiadBench and Omni-MATH problems (i.e., Olympiad-level ones). The agreement statistics in Table 2 further evidence that the more challenging problems usually need more annotators to achieve the annotation agreement, particularly for those samples with incorrect final answers. These results suggest the inherent challenge of our human annotation task.

3.3 Statistics

Table 2: Statistics of PROCESSBENCH. "% Process errors" denotes the proportion of samples with *erroneous reasoning steps* (i.e., annotated as erroneous) among all the samples with *correct final answers*. "% $\geq n$ steps" denotes the proportion of samples whose solutions have $\geq n$ steps (split by double line breaks). "% $3/n$ agreement" denotes the proportion of samples where the three-annotator agreement is achieved within n annotators, so $(\%\ 3/3) + (\%\ 3/4) + (\%\ 3/5) = 100\%$.

	GSM8K		**MATH**		**OlympiadBench**		**Omni-MATH**	
	error	**correct**	**error**	**correct**	**error**	**correct**	**error**	**correct**
# Samples	207	193	594	406	661	339	759	241
% Process errors (correct final answers)	$\frac{200-193}{200} = 3.5\%$		$\frac{500-406}{500} = 18.8\%$		$\frac{500-339}{500} = 32.2\%$		$\frac{500-241}{500} = 51.8\%$	
# Steps	5.3	5.1	6.8	6.0	8.9	8.7	8.6	7.4
% ≥ 5 steps	61.8%	57.5%	73.6%	70.4%	92.6%	92.3%	92.5%	81.7%
% ≥ 10 steps	3.4%	1.6%	17.8%	8.9%	33.9%	27.1%	29.2%	21.6%
% ≥ 15 steps	0.5%	0.0%	3.4%	2.0%	9.1%	8.8%	7.5%	4.1%
% 3/3 agreement	66.7%	95.9%	59.4%	91.9%	52.8%	85.0%	47.8%	80.1%
% 3/4 agreement	21.3%	3.6%	24.4%	4.7%	24.1%	9.1%	25.6%	13.7%
% 3/5 agreement	12.1%	0.5%	16.2%	3.4%	23.1%	5.9%	26.6%	6.2%

The resulting PROCESSBENCH has four subsets, consisting of 3,400 test cases in total. The detailed statistics are shown in Table 2 and Table 4 (in Appendix B), and we also plot in Figure 3 the distribution of error positions in erroneous samples. In general, the more challenging the problems, the more solution steps the models generate, and incorrect solutions usually contain more steps than correct ones. However, across all four subsets, a large proportion of errors occur in the earlier steps, such as steps 0-3 in GSM8K and MATH, and steps 1-5 in OlympiadBench and Omni-MATH.

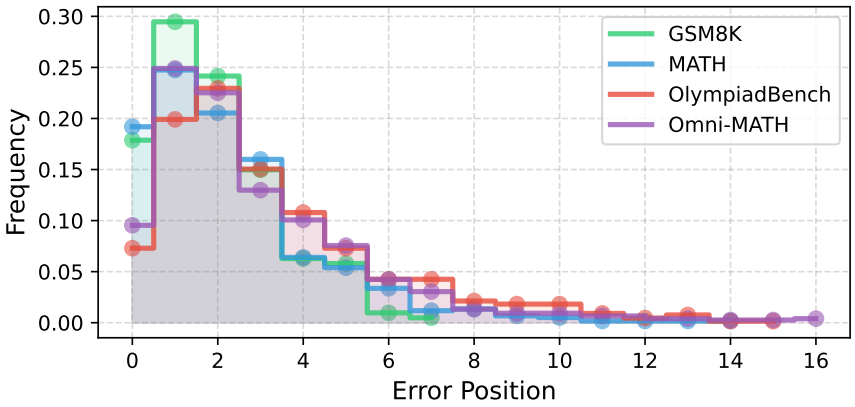

Figure 3: Distribution of error positions (indexed from 0; truncated to 16 for better visualization), corresponding to the *label* field as shown in Figure 2.

It is noteworthy that **while we have intentionally controlled an equal number of solutions with incorrect and correct final answers (200 each for GSM8K and 500 each for other subsets)**, the annotation

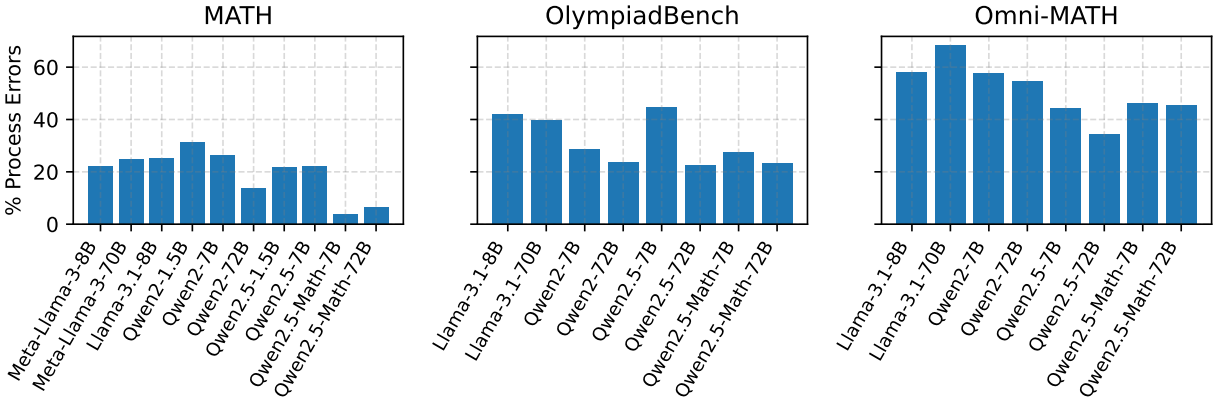

Figure 4: Process error ratios per models and subsets, computed as the proportions of samples annotated as *erroneous* among all the samples with *correct final answers* (same as in Table 2). The models used for solution generation slightly vary across different subsets, see Table 4 in Appendix B. We observe that no particular models have notably higher process error rates, while **the process error rates are consistently higher on more difficult problems for all the models.**

results reveal quite different numbers. Specifically, in the more challenging subsets like OlympiadBench and Omni-MATH, a larger proportion of solutions with correct final answers still contain erroneous steps. For instance, in OlympiadBench, $\frac{500-339}{500} = 32.2\%$ of solutions with correct final answers are found to contain process errors, while in Omni-MATH this proportion is even higher ($\frac{500-241}{500} = 51.8\%$). In contrast, these proportions in GSM8K and MATH are $\frac{200-193}{200} = 3.5\%$ and $\frac{500-406}{500} = 18.8\%$, respectively. In Figure 4, for each model used for solution generation, we plot the ratio of samples with *erroneous reasoning steps* (i.e., annotated as erroneous) among all the samples with *correct final answers*. We observe that the process error rates are consistently higher on more difficult problems. To our knowledge, our work is **the first to present evidence that on more challenging math problems, current language models are more prone to making process errors even when reaching correct final answers**. This also suggests the underlying limitation of rule-based RL in mathematical reasoning (i.e., rewarding merely according to the correctness of final answers) and further highlights the significance of identifying errors in the reasoning process.

4 Evaluation

4.1 Setup

For each subset of PROCESSBENCH, we calculate the accuracies on erroneous and correct samples, respectively, and additionally compute their harmonic mean as the **F1 score**. We primarily refer to F1 scores to compare model performance, as it balances model behaviors between being overly critical and being incapable of identifying errors.

We consider two types of models in the evaluation on PROCESSBENCH: *process reward models (PRMs)* and *critic models*.

Process Reward Models (PRMs) As a recently focal topic, PRMs are proposed to assess and supervise the intermediate steps in language models' reasoning process (Lightman et al., 2023), thus naturally falling in the scope of our research. In practice, PRMs are typically trained using the process labels for intermediate reasoning steps, outputting either the correctness prediction or a scalar score for each reasoning step during inference. Previous research usually evaluates PRMs based on their improvement in the Best-of-N (BoN) performance of another language model that generates solutions. However, this lacks a finer-grained inspection on their process assessment abilities, and the evaluation reliability can be heavily affected by the underlying solution generation model.

Our evaluation includes several open-source PRMs: (1) Math-Shepherd (Wang et al., 2024), which obtains the process label for each step via estimating the empirical probability of this step leading to the correct final answer. (2) Two LLaMA-3.1-based PRMs from Xiong et al. (2024b), which roughly follow the training methodology of Math-Shepherd but differ in the solution generation models and optimization objectives. (3) Two Qwen2.5-Math-based PRMs recently released by Skywork (2024). (4) We also train a PRM by fine-tuning Qwen2.5-Math-7B-Instruct on the PRM800K dataset, namely Qwen2.5-Math-7B-PRM800K. See Appendix C for its training details.

For the (1)(2)(4) PRMs, we extract the earliest erroneous step from their correctness predictions for reasoning steps. For the (3) PRMs, which produce scalar scores for each reasoning step, we first transform these scores into binary correctness predictions (using a threshold above which steps are considered as correct), and then extract the earliest erroneous step as we do for (1)(2)(4). The transformation threshold is determined as the one giving the highest F1 score on the GSM8K subset.

Critic Models Critic models aim to provide feedback and critique to model-generated texts, non-inclusively including verification, reflection, and correction or refinement. They have demonstrated promising utility in achieving scalable oversight (Saunders et al., 2022; McAleese et al., 2024).

Training critic models for specific domains typically requires significant and specialized effort, which is out of the scope of our work. Instead, we are more interested in the critique capabilities of *general language models*. The task definition (§ 3.1) of PROCESSBENCH enables us to apply simple prompt engineering to repurpose general language models as critic models. We show in Figure 7 in Appendix D the prompt template we implement for our evaluation. Specifically, models are prompted to return the index of the paragraph where the earliest error occurs as the *final answer*, similar to the conventional evaluation protocol for mathematical reasoning tasks (Cobbe et al., 2021; Hendrycks et al., 2021; Yang et al., 2024b).

Our evaluation includes the widely-used Qwen2 (Yang et al., 2024a), Qwen2.5 (Qwen, 2024a), Qwen2.5-Math (Yang et al., 2024b), Qwen2.5-Coder (Hui et al., 2024), and LLaMA-3 (Dubey et al., 2024) series open-source models, as well as the recently released QwQ-32B-Preview reasoning model (Qwen, 2024b). We also evaluate the proprietary GPT-4o (Hurst et al., 2024) and o1-mini (OpenAI, 2024) models. We report the performance of open-source models under majority voting over eight samplings, while we also report their performance under greedy decoding in Table 9 in Appendix E. For the proprietary model GPT-4o, we report the results under greedy decoding, while for o1-mini, we report the results under single sampling as its API does not support customized decoding parameters.

Table 3: Evaluation results on PROCESSBENCH. We report the F1 score of the respective accuracies on erroneous and correct samples. See Table 5 and Table 7 for breakdown of evaluation results.

Model	GSM8K	MATH	Olympiad-Bench	Omni-MATH	Average
*Open-source **Process Reward Models (PRMs)***					
Math-Shepherd-PRM-7B	47.9	29.5	24.8	23.8	31.5
RLHFlow-PRM-Mistral-8B	50.4	33.4	13.8	15.8	28.4
RLHFlow-PRM-Deepseek-8B	38.8	33.8	16.9	16.9	26.6
Skywork-PRM-1.5B	59.0	48.0	19.3	19.2	36.4
Skywork-PRM-7B	**70.8**	53.6	22.9	21.0	42.1
Qwen2.5-Math-7B-PRM800K (our trained)	68.2	**62.6**	**50.7**	**44.3**	**56.5**
*Open-source language models, prompted as **Critic Models***					
Meta-Llama-3-8B-Instruct	13.1	13.8	4.8	12.6	11.1
Meta-Llama-3-70B-Instruct	52.2	22.8	21.2	20.0	29.1
Llama-3.1-8B-Instruct	10.9	5.1	2.8	1.6	5.1
Llama-3.1-70B-Instruct	74.9	48.2	46.7	41.0	52.7
Llama-3.3-70B-Instruct	82.9	59.4	46.7	43.0	58.0
Qwen2.5-Math-7B-Instruct	26.8	25.7	14.2	12.7	19.9
Qwen2.5-Math-72B-Instruct	65.8	52.1	32.5	31.7	45.5
Qwen2.5-Coder-7B-Instruct	14.3	6.5	4.1	1.8	6.7
Qwen2.5-Coder-14B-Instruct	50.1	39.9	34.0	27.3	37.8
Qwen2.5-Coder-32B-Instruct	68.9	60.1	48.9	46.3	56.1
Qwen2-7B-Instruct	8.4	19.0	14.7	12.1	13.6
Qwen2-72B-Instruct	67.6	49.2	42.1	40.2	49.8
Qwen2.5-7B-Instruct	36.5	36.6	29.7	27.4	32.6
Qwen2.5-14B-Instruct	69.3	53.3	45.0	41.3	52.2
Qwen2.5-32B-Instruct	65.6	53.1	40.0	38.3	49.3
Qwen2.5-72B-Instruct	76.2	61.8	54.6	52.2	61.2
★ **QwQ-32B-Preview**	**88.0**	**78.7**	**57.8**	**61.3**	**71.5**
*Proprietary language models, prompted as **Critic Models***					
GPT-4o-0806	79.2	63.6	51.4	53.5	61.9
o1-mini	93.2	88.9	87.2	82.4	87.9

```
Alright, I've got this math problem to solve and a solution provided, but I need to review it
paragraph by paragraph to find any errors. The problem is to find the integer \(n\), where \(0
\leq n \leq 180\), such that \(\cos n^\circ = \cos 568^\circ\). Let's dive into the solution step
by step.

**Paragraph 0:**

The solution starts by recalling that the cosine function has a period of \(360^\circ\), which
means \(\cos \theta = \cos (\theta + 360k)\) for any integer \(k\). That's correct. Cosine repeats
every \(360^\circ\), so adding or subtracting multiples of \(360^\circ\) doesn't change the value
of the cosine.

**Paragraph 1:**

Given \(\cos 568^\circ\), the solution reduces this angle by subtracting \(360^\circ\) to get
\(208^\circ\), since \(568^\circ - 360^\circ = 208^\circ\). So, \(\cos 568^\circ = \cos
208^\circ\). That seems right. 208 degrees is within one cycle of \(0^\circ\) to \(360^\circ\).

**Paragraph 2:**

Now, the solution mentions that cosine is symmetric about \(180^\circ\), and states that \(\cos
\theta = \cos (180^\circ - \theta)\). Wait a minute, is that correct? Let me think about this.
Actually, cosine is symmetric about the y-axis, and the identity \(\cos \theta = \cos (-\theta)\)
reflects that. The identity \(\cos \theta = \cos (360^\circ - \theta)\) shows the symmetry about
\(360^\circ\), but symmetry about \(180^\circ\) isn't directly related to cosine's even property.

Hmm, maybe I need to recall some trigonometric identities here. Let's see, the cosine of an angle
in standard position is equal to the cosine of its reference angle, but with a sign depending on
the quadrant. 208 degrees is in the third quadrant, where cosine is negative.

Alternatively, there's the identity \(\cos (180^\circ + \theta) = -\cos \theta\), but that's not
directly applicable here.

Let me double-check the symmetry properties. Cosine is an even function, meaning \(\cos (-\theta)
= \cos \theta\), which shows symmetry about the y-axis. The identity \(\cos (360^\circ - \theta) =
\cos \theta\) shows symmetry every full cycle.

But the statement \(\cos \theta = \cos (180^\circ - \theta)\) isn't generally true. Let's test it
with a specific angle. For example, \(\theta = 30^\circ\): \(\cos 30^\circ = \sqrt{3}/2\), and
\(\cos (180^\circ - 30^\circ) = \cos 150^\circ = -\sqrt{3}/2\). These are not equal; in fact,
they're negatives of each other.

So, it seems there's an error in this paragraph. The correct identity involving \(180^\circ\) is
\(\cos (180^\circ - \theta) = -\cos \theta\), not \(\cos \theta\).

Therefore, the earliest error occurs in paragraph 2.

**Final Answer**

\[ \boxed{2} \]
```

Figure 5: Critique generated by QwQ-32B-Preview for the test case in Figure 2.

4.2 Results

We present the evaluation results in Table 3. Our observations are summarized as follows:

Generalization Across Difficulty From GSM8K and MATH to OlympiadBench and Omni-MATH, with the increased difficulty of math problems, we observe a consistent performance decline for all the models, which suggests the common challenge of both PRMs and critic models in generalization abilities.

Comparison Between PRMs and Critic Models We find that existing PRMs typically underperform the top prompt-driven critic models even on the simpler GSM8K and MATH subsets, suggesting that these PRMs struggle to indicate the correctness of the intermediate steps in mathematical reasoning. Moreover, when moving toward the more challenging OlympiadBench and Omni-MATH subsets, PRMs suffer from a more notable performance decline than critic models. This raises our concerns about the **generalization abilities and scalability of the current data synthesis methodologies used to build PRMs**. More specifically, current methodologies, as exemplified by Math-Shepherd (Wang et al., 2024), measure the correctness of an intermediate step by estimating the empirical probability of this step leading to the correct final answer. This kind of approach has two intuitive major issues: (1) The process labels heavily depend on the language model used to generate solutions (i.e., highly "on-policy"), which would naturally

fail to indicate the correctness of reasoning steps generated by other models. (2) As demonstrated in § 3.3, current language models are prone to making process errors even when reaching correct final answers. This could substantially invalidate the estimated process labels, particularly on the more challenging math problems. In contrast, **Qwen2.5-Math-7B-PRM800K**, which is straightforwardly fine-tuned on the *fully human-annotated* PRM800K training set, exhibits the significantly stronger performance and generalization ability than other PRMs.

Comparison Among Critic Models Compared to PRMs, critic models can benefit from separate reasoning processes when critiquing solutions, as they can "think" more before indicating the correctness of each solution step, which leads to their better performance in this error identification task. Within the same model family, the error identification performance favorably scales with increased model sizes. Notably, the recently released reasoning model **QwQ-32B-Preview performs best among the open-source models and is highly competitive with GPT-4o**. It is noteworthy that QwQ-32B-Preview achieves more *balanced accuracies on erroneous and correct samples* (see Table 5 and 7 in Appendix E). We show in Figure 5 an example of critique generated by QwQ-32B-Preview to the test case in Figure 2, which not only identifies the erroneous step but also provides the detailed thinking process and explanation. Nevertheless, QwQ-32B-Preview still lags behind o1-mini, suggesting that although the gap in problem-solving performance is getting closer between open-source and proprietary models, there still exists another large gap in their critique capabilities.

5 Conclusion

We introduce the PROCESSBENCH benchmark for measuring the ability to identify erroneous steps in mathematical reasoning, characterized by its high problem difficulty and solution diversity, large scale, rigorous human annotation, and simple evaluation protocol. Through extensive evaluation with existing process reward models (PRMs) and prompt-driven critic models, we draw two main observations: (1) Existing PRMs typically underperform critic models in identifying erroneous reasoning steps, and struggle more to generalize to challenging math problems. (2) Open-source language models, as exemplified by QwQ-32B-Preview, have demonstrated critique capabilities competitive with the proprietary model GPT-4o, yet still lag behind the reasoning-specialized o1-mini model. We envision PROCESSBENCH as a cornerstone testbed for advancing automated reasoning process assessment, a critical step toward achieving scalable oversight of language models.

Limitation Despite our best efforts throughout the entire benchmark construction process (§ 3.2), PROCESSBENCH may still contain inaccurate labels of error locations, particularly for the more challenging Olympiad-level math problems.

References

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

Samuel R Bowman, Jeeyoon Hyun, Ethan Perez, Edwin Chen, Craig Pettit, Scott Heiner, Kamilė Lukošiūtė, Amanda Askell, Andy Jones, Anna Chen, et al. Measuring progress on scalable oversight for large language models. *arXiv preprint arXiv:2211.03540*, 2022.

Boxi Cao, Keming Lu, Xinyu Lu, Jiawei Chen, Mengjie Ren, Hao Xiang, Peilin Liu, Yaojie Lu, Ben He, Xianpei Han, et al. Towards scalable automated alignment of llms: A survey. *arXiv preprint arXiv:2406.01252*, 2024.

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.

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

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

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

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.

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.

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

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Ryo Kamoi, Yusen Zhang, Nan Zhang, Jiawei Han, and Rui Zhang. When can LLMs actually correct their own mistakes? a critical survey of self-correction of LLMs. *Transactions of the Association for Computational Linguistics*, 12:1417–1440, 2024. doi: 10.1162/tacl_a_00713. URL https://aclanthology.org/2024.tacl-1.78.

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.

Zicheng Lin, Zhibin Gou, Tian Liang, Ruilin Luo, Haowei Liu, and Yujiu Yang. Criticbench: Benchmarking llms for critique-correct reasoning. *arXiv preprint arXiv:2402.14809*, 2024.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, Jiao Sun, and Abhinav Rastogi. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592*, 2024.

Nat McAleese, Rai Michael Pokorny, Juan Felipe Ceron Uribe, Evgenia Nitishinskaya, Maja Trebacz, and Jan Leike. Llm critics help catch llm bugs. *arXiv preprint arXiv:2407.00215*, 2024.

OpenAI. Openai o1-mini: Advancing cost-efficient reasoning, 2024. URL https://openai.com/index/openai-o1-mini-advancing-cost-efficient-reasoning/.

Team Qwen. Qwen2.5: A party of foundation models, September 2024a. URL https://qwenlm.github.io/blog/qwen2.5/.

Team Qwen. Qwq: Reflect deeply on the boundaries of the unknown, November 2024b. URL https://qwenlm.github.io/blog/qwq-32b-preview/.

William Saunders, Catherine Yeh, Jeff Wu, Steven Bills, Long Ouyang, Jonathan Ward, and Jan Leike. Self-critiquing models for assisting human evaluators. *arXiv preprint arXiv:2206.05802*, 2022.

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

Alan Wake, Albert Wang, Bei Chen, CX Lv, Chao Li, Chengen Huang, Chenglin Cai, Chujie Zheng, Daniel Cooper, Ethan Dai, et al. Yi-lightning technical report. *arXiv preprint arXiv:2412.01253*, 2024.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 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, August 2024. doi: 10.18653/v1/2024.acl-long.510. URL https://aclanthology.org/2024.acl-long.510.

Wei Xiong, Chengshuai Shi, Jiaming Shen, Aviv Rosenberg, Zhen Qin, Daniele Calandriello, Misha Khalman, Rishabh Joshi, Bilal Piot, Mohammad Saleh, et al. Building math agents with multi-turn iterative preference learning. *arXiv preprint arXiv:2409.02392*, 2024a.

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

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024a.

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

Zihao Zhou, Shudong Liu, Maizhen Ning, Wei Liu, Jindong Wang, Derek F Wong, Xiaowei Huang, Qiufeng Wang, and Kaizhu Huang. Is your model really a good math reasoner? evaluating mathematical reasoning with checklist. *arXiv preprint arXiv:2407.08733*, 2024.

A Example of Solution Reformatting

```
The given equation is of an ellipse in the standard form

\[
\frac{(x-h)^2}{a^2} + \frac{(y-k)^2}{b^2} = 1,
\]

where $(h, k)$ is the center of the ellipse, $a$ is the semi-
major axis length, and $b$ is the semi-minor axis length.

In this case, the center of the ellipse is $(6, 3)$, $a = 5$,
and $b = 3$.

The foci of an ellipse are located along the major axis, which
is the line segment connecting the two vertices of the ellipse.
The distance from the center of the ellipse to each focus is
given by $\sqrt{a^2 - b^2}$.

First, we calculate the distance from the center to each focus:

\[
\sqrt{a^2 - b^2} = \sqrt{5^2 - 3^2} = \sqrt{25 - 9} = \sqrt{16}
= 4.
\]

Since the major axis of the ellipse is horizontal (because $a
> b$), the foci are located at a distance of 4 units to the
left and right of the center of the ellipse.

Therefore, the two foci are at $(6 \pm 4, 3) = (2, 3)$ and
$(10, 3)$.

The one with the larger $x$-coordinate is $(10, 3)$.

The answer is: $\boxed{(10, 3)}$.
```

⇨

```
The given equation is of an ellipse in
the standard form \[\frac{(x-h)^2}{a^2} +
\frac{(y-k)^2}{b^2} = 1,\] where \((h,
k)\) is the center of the ellipse, \(a\)
is the semi-major axis length, and \(b\)
is the semi-minor axis length. In this
case, the center of the ellipse is \((6,
3)\), \(a = 5\), and \(b = 3\).

The foci of an ellipse are located along
the major axis, which is the line segment
connecting the two vertices of the
ellipse. The distance from the center of
the ellipse to each focus is given by
\(\sqrt{a^2 - b^2}\). First, we calculate
the distance from the center to each
focus: \[\sqrt{a^2 - b^2} = \sqrt{5^2 -
3^2} = \sqrt{25 - 9} = \sqrt{16} = 4.\]

Since the major axis of the ellipse is
horizontal (because \(a > b\)), the foci
are located at a distance of 4 units to
the left and right of the center of the
ellipse. Therefore, the two foci are at
\((6 \pm 4, 3) = (2, 3)\) and \((10, 3)\).

The one with the larger \(x\)-coordinate
is \((10, 3)\). The answer is:
\(\boxed{(10, 3)}\).
```

Figure 6: Example of solution reformatting. The left is the original solution (generated by Qwen2-7B-Instruct) and the right is the reformatted one. The problem, coming from the MATH test set, is "*The ellipse* $\frac{(x-6)^2}{25} + \frac{(y-3)^2}{9} = 1$ *has two foci. Find the one with the larger x-coordinate. Enter your answer as an ordered pair, like* $(2, 1)$."

B Breakdown Statistics of PROCESSBENCH

Table 4: Breakdown statistics of PROCESSBENCH. †: We encountered a code bug when using Llama-3.1-70B-Instruct and Qwen2.5-72B-Instruct to generate solutions for the MATH problems, thus their counts are all zero in the MATH subset of PROCESSBENCH. ‡: For the more challenging OlympiadBench and Omni-MATH problems, we exclude models with lower accuracies from subsequent annotation.

Generator	GSM8K		MATH†		OlympiadBench‡		Omni-MATH‡	
	error	correct	error	correct	error	correct	error	correct
Meta-Llama-3-8B-Instruct	11	13	56	14	0	0	0	0
Meta-Llama-3-70B-Instruct	16	15	92	49	0	0	0	0
Llama-3.1-8B-Instruct	38	23	86	53	116	48	131	31
Llama-3.1-70B-Instruct	7	28	0	0	85	32	103	19
Qwen2-1.5B-Instruct	37	4	36	11	0	0	0	0
Qwen2-7B-Instruct	31	21	89	42	63	45	96	35
Qwen2-72B-Instruct	9	11	56	51	64	48	71	25
Qwen2.5-1.5B-Instruct	32	10	31	43	0	0	0	0
Qwen2.5-7B-Instruct	12	15	62	35	86	37	75	29
Qwen2.5-72B-Instruct	2	21	0	0	67	38	88	38
Qwen2.5-Math-7B-Instruct	8	14	47	49	99	48	103	29
Qwen2.5-Math-72B-Instruct	4	18	39	59	81	43	92	35
Total	207	193	594	406	661	339	759	241
	400		1,000		1,000		1,000	

C Training Details of Qwen2.5-Math-7B-PRM800K

Qwen2.5-Math-7B-PRM800K is obtained by fine-tuning Qwen2.5-Math-7B-Instruct on the PRM800K training set. We replace the original language modeling head with a new reward modeling head that outputs binary classification logits. The classification loss is computed at the second line break positions in all the "\n\n". We treat the original 1 and 0 labels in PRM800K as our positive labels, while -1 as negative ones. To eliminate test data contamination, we also remove the PRM800K training samples that have the same problems in PROCESSBENCH.

D Prompt Template for Critic Model Evaluation

```
The following is a math problem and a solution (split into paragraphs, enclosed with tags and
indexed from 0):

[Math Problem]

...(math problem)...

[Solution]

<paragraph_0>
...(paragraph 0 of solution)...
</paragraph_0>

...

<paragraph_n-1>
...(paragraph n-1 of solution)...
</paragraph_n-1>

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{}.
```

Figure 7: Prompt template for critic model evaluation. The **blue texts** indicate the input math problem and the solution (split into paragraphs). The **red texts** describe the required output content and format.

E Supplementary Evaluation Results

Table 5: Breakdown of evaluation results on the GSM8K and MATH subsets of PROCESSBENCH. The open-source language models (middle block) are evaluated via *majority voting* over eight samplings.

Model	GSM8K			MATH		
	error	correct	F1	error	correct	F1
*Open-source **Process Reward Models (PRMs)***						
Math-Shepherd-PRM-7B	32.4	91.7	47.9	18.0	82.0	29.5
RLHFlow-PRM-Mistral-8B	33.8	99.0	50.4	21.7	72.2	33.4
RLHFlow-PRM-Deepseek-8B	24.2	98.4	38.8	21.4	80.0	33.8
Skywork-PRM-1.5B	50.2	71.5	59.0	37.9	65.3	48.0
Skywork-PRM-7B	61.8	82.9	**70.8**	43.8	69.2	53.6
Qwen2.5-Math-7B-PRM800K (our trained)	53.1	95.3	68.2	48.0	90.1	**62.6**
*Open-source language models, prompted as **Critic Models***						
Meta-Llama-3-8B-Instruct	42.5	7.8	13.1	28.6	9.1	13.8
Meta-Llama-3-70B-Instruct	35.7	96.9	52.2	13.0	93.3	22.8
Llama-3.1-8B-Instruct	44.4	6.2	10.9	41.9	2.7	5.1
Llama-3.1-70B-Instruct	64.3	89.6	74.9	35.4	75.6	48.2
Llama-3.3-70B-Instruct	72.5	96.9	82.9	43.3	94.6	59.4
Qwen2.5-Math-7B-Instruct	15.5	100.0	26.8	14.8	96.8	25.7
Qwen2.5-Math-72B-Instruct	49.8	96.9	65.8	36.0	94.3	52.1
Qwen2.5-Coder-7B-Instruct	7.7	100.0	14.3	3.4	98.3	6.5
Qwen2.5-Coder-14B-Instruct	33.8	96.4	50.1	25.4	92.4	39.9
Qwen2.5-Coder-32B-Instruct	54.1	94.8	68.9	44.9	90.6	60.1
Qwen2-7B-Instruct	40.6	4.7	8.4	30.5	13.8	19.0
Qwen2-72B-Instruct	57.0	82.9	67.6	37.7	70.9	49.2
Qwen2.5-7B-Instruct	40.6	33.2	36.5	30.8	45.1	36.6
Qwen2.5-14B-Instruct	54.6	94.8	69.3	38.4	87.4	53.3
Qwen2.5-32B-Instruct	49.3	97.9	65.6	36.7	95.8	53.1
Qwen2.5-72B-Instruct	62.8	96.9	76.2	46.3	93.1	61.8
QwQ-32B-Preview	81.6	95.3	**88.0**	78.1	79.3	**78.7**
*Proprietary language models, prompted as **Critic Models***						
GPT-4o-0806	70.0	91.2	79.2	54.4	76.6	63.6
o1-mini	88.9	97.9	93.2	83.5	95.1	88.9

Table 6: For the two PRMs from Skywork (2024), we additionally adjust the threshold (§ 4.1) as the one leading to the highest F1 score on each subset (i.e., each subset adopts a respective optimal threshold), which can be viewed as the two PRMs' *upper bound* performance on PROCESSBENCH. This table presents the results on the GSM8K and MATH subsets, which are marginally higher than those in Table 5 that all adopt the threshold selected on the GSM8K subset.

Model	GSM8K			MATH		
	error	correct	F1	error	correct	F1
Skywork-PRM-1.5B (respective thresholds)	50.2	71.5	59.0	38.2	70.4	49.5
Skywork-PRM-7B (respective thresholds)	61.8	82.9	70.8	44.1	70.9	54.4

Table 7: Breakdown of evaluation results on the OlympiadBench and Omni-MATH subsets of PROCESSBENCH. The open-source language models (middle block) are evaluated via *majority voting* over eight samplings.

Model	OlympiadBench			Omni-MATH		
	error	correct	F1	error	correct	F1
*Open-source **Process Reward Models (PRMs)***						
Math-Shepherd-PRM-7B	15.0	71.1	24.8	14.2	73.0	23.8
RLHFlow-PRM-Mistral-8B	8.2	43.1	13.8	9.6	45.2	15.8
RLHFlow-PRM-Deepseek-8B	10.1	51.0	16.9	10.1	51.9	16.9
Skywork-PRM-1.5B	15.4	26.0	19.3	13.6	32.8	19.2
Skywork-PRM-7B	17.9	31.9	22.9	14.0	41.9	21.0
Qwen2.5-Math-7B-PRM800K (our trained)	35.7	87.3	**50.7**	29.8	86.3	**44.3**
*Open-source language models, prompted as **Critic Models***						
Meta-Llama-3-8B-Instruct	27.1	2.7	4.8	26.1	8.3	12.6
Meta-Llama-3-70B-Instruct	12.0	92.0	21.2	11.2	91.7	20.0
Llama-3.1-8B-Instruct	32.4	1.5	2.8	32.0	0.8	1.6
Llama-3.1-70B-Instruct	35.1	69.9	46.7	30.7	61.8	41.0
Llama-3.3-70B-Instruct	31.0	94.1	46.7	28.2	90.5	43.0
Qwen2.5-Math-7B-Instruct	7.7	91.7	14.2	6.9	88.0	12.7
Qwen2.5-Math-72B-Instruct	19.5	97.3	32.5	19.0	96.3	31.7
Qwen2.5-Coder-7B-Instruct	2.1	99.1	4.1	0.9	98.3	1.8
Qwen2.5-Coder-14B-Instruct	20.7	94.1	34.0	15.9	94.2	27.3
Qwen2.5-Coder-32B-Instruct	33.4	91.2	48.9	31.5	87.6	46.3
Qwen2-7B-Instruct	22.4	10.9	14.7	20.0	8.7	12.1
Qwen2-72B-Instruct	34.0	55.2	42.1	32.3	53.1	40.2
Qwen2.5-7B-Instruct	26.5	33.9	29.7	26.2	28.6	27.4
Qwen2.5-14B-Instruct	31.5	78.8	45.0	28.3	76.3	41.3
Qwen2.5-32B-Instruct	25.3	95.9	40.0	24.1	92.5	38.3
Qwen2.5-72B-Instruct	38.7	92.6	54.6	36.6	90.9	52.2
QwQ-32B-Preview	61.4	54.6	**57.8**	55.7	68.0	**61.3**
*Proprietary language models, prompted as **Critic Models***						
GPT-4o-0806	45.8	58.4	51.4	45.2	65.6	53.5
o1-mini	80.2	95.6	87.2	74.8	91.7	82.4

Table 8: For the two PRMs from Skywork (2024), we additionally adjust the threshold (§ 4.1) as the one leading to the highest F1 score on each subset (i.e., each subset adopts a respective optimal threshold), which can be viewed as the two PRMs' *upper bound* performance on PROCESSBENCH. This table presents the results on the OlympiadBench and Omni-MATH subsets, which are slightly higher than those in Table 7 that all adopt the threshold selected on the GSM8K subset.

Model	OlympiadBench			Omni-MATH		
	error	correct	F1	error	correct	F1
Skywork-PRM-1.5B (respective thresholds)	15.3	47.5	23.1	14.0	58.5	22.6
Skywork-PRM-7B (respective thresholds)	18.9	48.1	27.1	14.4	58.1	23.1

Table 9: Breakdown of evaluation results of the open-source language models (prompted as critic models) using *greedy decoding*.

Model	GSM8K			MATH		
	error	correct	F1	error	correct	F1
Meta-Llama-3-8B-Instruct	28.5	9.3	14.1	20.9	5.7	8.9
Meta-Llama-3-70B-Instruct	39.6	93.8	55.7	21.9	72.2	33.6
Llama-3.1-8B-Instruct	36.7	17.1	23.3	23.6	7.9	11.8
Llama-3.1-70B-Instruct	57.5	77.7	66.1	37.7	53.9	44.4
Llama-3.3-70B-Instruct	66.2	96.9	78.6	38.4	93.1	54.4
Qwen2.5-Math-7B-Instruct	14.5	99.0	25.3	13.1	94.8	23.1
Qwen2.5-Math-72B-Instruct	45.9	96.4	62.2	34.3	94.6	50.4
Qwen2.5-Coder-7B-Instruct	0.0	20.2	0.0	0.2	25.6	0.3
Qwen2.5-Coder-14B-Instruct	20.3	99.0	33.7	15.2	96.1	26.2
Qwen2.5-Coder-32B-Instruct	50.7	93.8	65.8	39.7	88.2	54.8
Qwen2-7B-Instruct	28.0	0.0	0.0	19.0	5.2	8.1
Qwen2-72B-Instruct	56.5	82.4	67.0	35.5	66.7	46.4
Qwen2.5-7B-Instruct	36.7	66.3	47.3	23.7	63.8	34.6
Qwen2.5-14B-Instruct	47.8	93.8	63.3	40.4	86.9	55.2
Qwen2.5-32B-Instruct	43.0	97.9	59.8	33.3	95.6	49.4
Qwen2.5-72B-Instruct	61.4	98.4	75.6	45.3	91.9	60.7
QwQ-32B-Preview	74.9	67.4	70.9	58.6	54.2	56.3

Model	OlympiadBench			Omni-MATH		
	error	correct	F1	error	correct	F1
Meta-Llama-3-8B-Instruct	17.2	0.6	1.1	17.3	4.1	6.7
Meta-Llama-3-70B-Instruct	20.9	41.6	27.8	20.9	50.2	29.6
Llama-3.1-8B-Instruct	19.1	5.6	8.7	17.1	10.0	12.6
Llama-3.1-70B-Instruct	32.8	32.4	32.6	29.5	39.0	33.6
Llama-3.3-70B-Instruct	30.9	90.0	46.0	27.1	86.3	41.3
Qwen2.5-Math-7B-Instruct	6.4	79.1	11.8	4.7	78.0	8.9
Qwen2.5-Math-72B-Instruct	17.2	95.0	29.2	18.3	93.4	30.6
Qwen2.5-Coder-7B-Instruct	0.0	13.3	0.0	0.0	27.8	0.0
Qwen2.5-Coder-14B-Instruct	9.1	95.6	16.6	6.2	95.9	11.6
Qwen2.5-Coder-32B-Instruct	31.8	86.7	46.5	31.5	84.6	45.9
Qwen2-7B-Instruct	14.1	2.9	4.9	13.7	2.9	4.8
Qwen2-72B-Instruct	33.4	48.1	39.4	30.4	48.1	37.3
Qwen2.5-7B-Instruct	25.4	46.0	32.7	26.1	43.6	32.6
Qwen2.5-14B-Instruct	30.9	76.4	44.0	27.0	72.6	39.4
Qwen2.5-32B-Instruct	22.4	90.0	35.9	22.4	87.6	35.7
Qwen2.5-72B-Instruct	33.7	88.5	48.9	33.7	88.4	48.8
QwQ-32B-Preview	37.8	31.9	34.6	29.5	41.9	34.6