A. Related Work

A.1. Multi-Modal Benchmarks

With the fast development of recent MLLMs [1, 3, 21–23], it becomes essential to evaluate their performance across a variety of tasks to understand their strengths and weaknesses, which is crucial for guiding future developments and enhancements [5, 24, 26, 28–37]. For example, MMMU [5] has been proposed to evaluate the scientific problem-solving abilities of MLLMs across a range of college-level subjects. MathVista [24] is widely used to assess the mathematical capabilities of MLLMs. While considerable progress has been achieved, these benchmarks often focus solely on evaluating final answers, overlooking crucial information of intermediate reasoning steps and potentially leading to unreliable results.

A.2. MLLM-based Process Judges

As MLLMs regularly make mistakes when solving scientific problems [5, 18], evaluating the validity of their reasoning processes is critical for ensuring reliability and uncovering fine-grained model weaknesses. Since human evaluation is costly and time-consuming, prompting MLLMs as automated process judges has become a common practice. MLLM-based process judges have been widely used to automatically evaluate reasoning steps of multi-modal language models. For example, MM-Math [17] incorporates MLLM-as-a-judge to automatically analyze solution steps, identifying and categorizing errors into specific error types. OlympicArena [8] employs GPT-4V to conduct process-level evaluations, scoring the correctness of each reasoning step to ensure a rigorous assessment. MathVerse [18] introduces a Chain-of-Thought (CoT) evaluation strategy, using GPT-4V to extract and assess key reasoning steps, providing fine-grained error analysis and nuanced scoring that goes beyond binary correctness. MME-CoT [19] extends this approach by evaluating CoT reasoning across multiple domains, while introducing metrics for reasoning quality, robustness, and efficiency.

Despite the widespread adoption, reliability of MLLM-based judges themselves is rarely scrutinized. Besides, current evaluation predominantly relies on proprietary models, suffering from prohibitive costs and reproducibility instability. This necessitates the development of open-source process judges. To address these challenges, we introduce **ProJudgeBench**, a comprehensive benchmark for assessing MLLMs' capabilities as process judges, and **ProJudge-173k**, a large-scale instruction-tuning dataset designed to enhance open-source MLLMs' process evaluation abilities.

A.3. Process Evaluation Benchmarks

There exist several benchmarks related to assessing process evaluation abilities of LLMs. MR-GSM8K [12] introduces a meta-reasoning paradigm, requiring LLMs to transition from solving problems to evaluating the correctness of reasoning steps. MathCheck-GSM [14] presents a checklist-based framework, where LLMs are tasked with evaluating both final answers and intermediate reasoning steps. CriticBench [13] assesses the ability of LLMs to critique and correct their reasoning across multiple domains. ProcessBench [15] measures the ability to identify erroneous steps in mathematical reasoning, particularly in competition-level problems. PRMBench [16] synthesize erroneous steps based on PRM800k [11], evaluating fine-grained error detection capabilities of PRMs across multiple dimensions.

Our ProJudgeBench is distinguished from prior benchmarks in three key aspects: First, it covers four scientific discipline with multi-modal content and varying difficutly levels, reflecting the complexity of real-world reasoning tasks. Second, test cases are curated from diverse model-generated solutions instead of synthetic data, capturing broad realistic reasoning behaviors and error patterns. Third, each step is human-annotated for correctness, error type and explanation, enabling fine-grained evaluation of models' error-diagnosing capabilities.

B. Definitions of Error Types

As described in Section 3.1, we define seven error types based on a comprehensive analysis of common mistakes that models tend to make during long-chain reasoning processes. The definitions of each category are displayed in Table 5.

C. Detailed Information of ProJudgeBench

In this section, we provide detailed information on ProJudgeBench, including the list of MLLMs used for generating solutions in data construction, instruction for human annotators, our annotation website, quality control process, and breakdown statistics.

C.1. Data Construction

As described in Section 3.2, we utilize 10 distinct MLLMs to generate solutions in ProJudgeBench. The list of utilized MLLMs are presented in Table 6.

Error Types	**Definitions**
Numerical Calculation Error	Errors in basic arithmetic operations such as addition, subtraction, division, or square roots.
Symbolic Calculation Error	Errors in manipulating algebraic expressions, such as incorrect expansion, factoring, simplification, or solving equations with variables.
Visual Interpretation Error	Errors in interpreting graphical data, such as misidentifying coordinates, shapes, spatial relationships, or data within figures.
Reasoning Error	Errors in the logical thinking process that lead to incorrect conclusions, such as flawed arguments, invalid inferences, or gaps in the logical flow of the solution.
Knowledge Error	Errors caused by insufficient understanding or incorrect application of necessary knowledge (e.g., concepts, formulas, theorems, methods), or using outdated or incorrect information.
Question Understanding Error	Errors due to misunderstanding or misinterpreting the problems' conditions or requirements, such as misreading questions or misapplying given conditions.
No solution provided	The model refuses to answer, fails to follow instructions to make a solution, or encounters anomalies in generation process such as repetitive responses or incomplete outputs.

Table 5. Definitions of seven error types.

MLLMs as solution generators in ProJudgeBench			
InternVL2.5-8B InternVL2.5-26B InternVL2.5-38B	Qwen2.5-VL-Instruct-3B Qwen2.5-VL-Instruct-7B Qwen2.5-VL-Instruct-72B	MiniCPM-V-2_6 (8B) QVQ-72B-Preview	LLaVA-OneVision (7B) GPT-4o

Table 6. List of MLLMs used in ProJudgeBench to generate solutions.

C.2. Quality Control

To ensure annotation reliability, we implement a rigorous quality control mechanism. First, a random subset of solutions are selected for repeated annotation to measure inter-annotator agreement. In cases where annotations exhibit significant discrepancies, the solutions are flagged for re-annotation. Additionally, solutions with missing annotations or contradictory annotations (e.g., a step marked as correct but assigned an error type) are also flagged for review and re-annotation. Besides, we conduct regular annotation review meetings where annotators discuss challenging cases and resolve ambiguities collaboratively. We also develop a detailed annotation guideline document, which is continuously updated based on annotator feedback and edge cases encountered during the annotating process. The multi-layered approach ensures a high level of consistency and reliability in the final annotations.

C.3. BreakDown Statistics

The breakdown statistics of ProJudgeBench is shown in Table 7. We can see that, as the difficulty of the problems increases, the average number of steps per solution also rises. Besides, higher-difficulty problems exhibit a greater proportion of erroneous steps and more diverse error types, with competition-level math problems reaching up to 512 steps, indicating that models tend to generate more steps when tackling complex problems. Higher-difficulty problems also exhibit a greater proportion of erroneous steps and more diverse error types, with some solutions containing up to 8 erroneous steps and 4 distinct error types. This highlights the importance of fine-grained process evaluation: if the judge model can diagnose all these errors, it will enable a more comprehensive analysis of the model's weaknesses, offering targeted feedback for improvement.

We also plot the frequency of the first occurrence step for different error types in Figure 7 . While the distribution varies across error categories, a consistent overall pattern emerges: errors peak in frequency during the earlier steps (e.g., steps 0-3)

	Overall	Math		Physics		Chemistry		Biology	
		K12	Comp	K12	Comp	K12	Comp	K12	Comp
# Samples	2400	450	150	300	300	300	300	300	300
Avg. Steps	20.8	23.7	21.3	21.5	26.0	20.0	19.9	15.2	18.0
Max. Steps	470	470	215	199	353	209	197	78	150
% Error Steps	21.6	23.0	19.7	21.2	23.1	18.9	24.4	15.7	23.4
Avg. Error Steps	6.6	8.1	7.0	7.0	8.4	5.7	6.2	4.2	5.7
Max. Error Types	5	4	4	4	4	4	5	3	4

Table 7. Statistics of ProJudgeBench. K12 and Comp represent normal and competition-level problems, respectively.

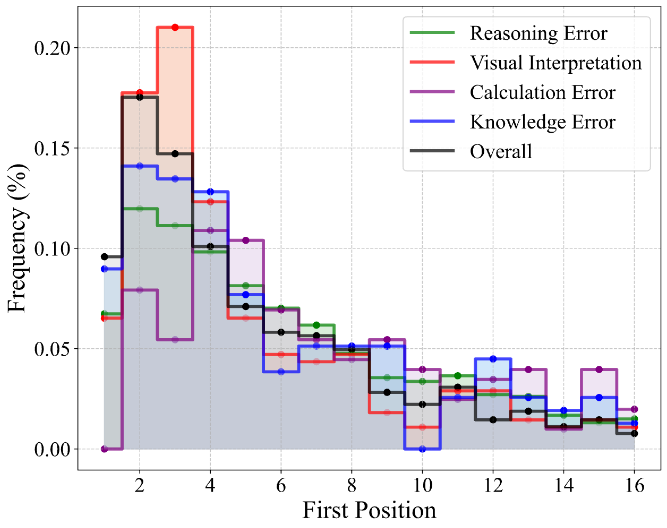

Figure 7. Distribution of the first occurance steps for different error types. Truncated to 16 for better visualization.

and gradually decline thereafter. Visual interpretation errors show a distinct peak at step 3, highlighting the model's initial struggles with accurately interpreting visual elements. Calculation errors, on the other hand, are more evenly distributed across all steps, indicating persistent challenges in performing precise mathematical operations throughout the reasoning process. In contrast, reasoning errors become more frequent in later steps, suggesting that models face increasing difficulty in maintaining logical consistency as the solution progresses.

D. Detailed Information of ProJudge-173k

In this section, we provide detailed information on ProJudge-173k, including the list of MLLMs used for generating solutions in data construction, data filtering process, and breakdown statistics.

D.1. Data Construction

As described in Section 4.1, we utilize 9 distinct MLLMs to generate solutions in ProJudge-173k. The list of utilized MLLMs are presented in Table 8.

MLLMs as solution generators in ProJudge-173k		
InternVL2.5-8B	Qwen2.5-VL-Instruct-3B	MiniCPM-V-2_6 (8B)
InternVL2.5-26B	Qwen2.5-VL-Instruct-7B	QVQ-72B-Preview
InternVL2.5-38B	Qwen2.5-VL-Instruct-72B	LLaVA-OneVision (7B)

Table 8. List of MLLMs used in ProJudge-173k to generate solutions.

	Total	Camel-AI	K12	Olympiad
# Samples	173,354	26,249	93,184	53,921
# Problems	26,084	8,000	12,000	6,084
# Math	58,922	7,196	15,908	35,818
# Physics	40,910	6,268	16,539	18,103
# Chemistry	37,789	6,656	31,133	0
# Biology	35,733	6,129	29,604	0
Avg. Steps	18	11	17	22
Max. Steps	926	77	487	926
% Error	24.72	20.	23.55	27.52

Table 9. Statistics of data sources and quantities in ProJudge-173k.

D.2. Data Filtering

As described in Section 4.1, we conduct rigorous filtering processes to ensure data quality. Specifically, we apply three quality control mechanisms: (1) **Format Consistency**: We remove samples that deviate from the predefined format, where each step in the student solution must be annotated as a tuple containing the step description, correctness, error type, and a brief explanation. Additionally, samples with mismatched step counts between annotations and student solutions are discarded. (2) **Annotation Consistency**: Samples with contradictory or incomplete annotations, such as steps marked as incorrect but lacking error type or cause descriptions, are excluded. (3) **Error Coverage**: Samples with insufficient error diversity or repetitive error patterns are excluded to maintain dataset variety.

D.3. BreakDown Statistics

The statistics of the data sources and quantities are presented in Table 9. By providing detailed error analysis, realistic reasoning paths, and diverse problem types, our dataset lays a solid foundation for advancing research in process evaluation, particularly for improving model's capabilities of error diagnosis in complex reasoning tasks.

E. Detailed Information of Process Evaluation

E.1. Fine-tuning Details

In the experiments, we fine-tune InternVL2.5-8B, Qwen2.5-VL-3B-instruct and Qwen2.5-VL-7B-Instruct on ProJudge-173k with Dynamic Dual-Phase strategy. The training is conducted on 8 H100 GPUS. All the models are fine-tuned using LoRA for one epoch, with the same training set. The global batch size is set to 16, with per-device batch size of 4 and gradient accumulation steps of 2. For InternVL2.5-8B, we employ a learning rate of 4e-5, optimized using a cosine learning rate scheduler with a warmup ratio of 0.03. Additionally, we applied weight decay of 0.05 to regularize the training process and prevent overfitting. For Qwen2.5-VL-3B and Qwen2.5-VL-7B, the models are trained with a learning rate of 1.0e-4, also using a cosine learning rate scheduler and a warmup ratio of 0.1. Both fine-tuning processes utilize mixed-precision training (bf16) to accelerate computation and reduce memory usage. For InternVL2.5-8B, we additionally enable gradient checkpointing to further optimize memory usage during training.

We conduct hyperparameter tuning for the switching probability p in Table 10, and set $p = 0.2$ for optimal performance.

p	0.1	0.2	0.3	0.4	0.5
InternVL2.5-8B	82.55	**84.50**	83.78	83.65	83.26
Qwen2.5-VL-Instruct-7B	83.65	**83.72**	80.47	81.05	80.34

Table 10. Effect of p on step-level error detection accuracy.

F. Prompts

F.1. Prompts for Injection Errors

As discussed in Section 4.1, we utilize GPT-4o to intentionally inject errors into correct solutions. The prompts we use are displayed in Table 11.

---

1. System Prompt

You are a highly experienced educator with a strong understanding of both solving problems and mimicking realistic, common mistakes made by students or AI.

2. User Content

Task: Analyse the following question, tested knowledge points and reference step-by-step solution.
Introduce REASONABLE and REALISTIC errors into solution that mimic COMMON mistakes made by students or AI.

Instructions:
1. Understand the following error categories and incorporate one or more of the following error types:
a. Numerical Calculation Error. Errors in basic arithmetic operations such as addition, subtraction, division, or square roots.
b. Symbolic Calculation Error. Errors in manipulating algebraic expressions, such as incorrect expansion, factoring, simplification, or solving equations with variables.
c. Visual Interpretation Error. Errors in interpreting graphical data, such as misidentifying coordinates, shapes, spatial relationships, or data within figures.
d. Reasoning Error. Errors in the logical thinking process that lead to incorrect conclusions, such as flawed arguments, invalid inferences, or gaps in the logical flow of the solution.
e. Knowledge Error. Errors caused by insufficient understanding or incorrect application of necessary knowledge (e.g., concepts, formulas, theorems, methods), or using outdated or incorrect information.
f. Question Understanding Error. Errors due to misunderstanding or misinterpreting the problems' conditions or requirements, such as misreading questions or misapplying given conditions.

2. Introduce Errors:
a. Insert multiple errors into the reference solution steps.
b. Errors should be RESONABLE, REALISTIC and resemble COMMON mistakes, NOT arbitrary or overly obvious.
c. AVOID REPEATING the same error reason across steps. ENSURE each step is evaluated INDIVIDUALLT.

3. Generate Erroneous Solutions:
a. Provide 3–5 erroneous solutions to cover diverse possibilities.
b. Ensure the erroneous solutions align with the question, remain logically consistent and misleading enough to challenge the reader.

4. Response Format: Present each erroneous solution step-by-step in the following format.
a. Mark each step as 1 (Correct) or 0 (Incorrect).
b. For incorrect steps, specify the error type and a brief error description.
c. Example:
```
[
    [
        ["Step Description 1", 0, "Error category","a brief error descrition"],
        ["Step Description 2", 1, "",""],
    ],
]
```
d. Strictly adhere to the format! DO NOT add any explanations, extra content, or annotations outside the specified format!

Table 11. Prompt for injecting errors.

F.2. Prompts for Solution Generation

As described in Section 3.2 and Section 4.1, we use diverse MLLMs to generate solutions, capturing a wide range of realistic reasoning behaviors and error patterns. The prompts we use for generating solutions are displayed in Table 12.

---

1. System Prompt

You are a highly skilled student proficient in solving scientific problems.

2. User Content

Based on the given images, solve the following question.

Here is some context information for this question, which might assist you in solving it: {context}*

Problem: {problem}

Think step by step logically, considering all relevant information before answering.
Write out the solution process, and use the same LaTeX format as the question in the solution process.
Please end your response with: ”The final answer is $\boxed{ANSWER}$.”

Table 12. Prompt for generating solutions.

.

F.3. Prompts for Spliting Solutions into Steps

To standardize the step granularity of each solution, we prompt Qwen2.5-72B-Instruct to split solutions into logically complete and progressive steps. The prompts we use for splitting are displayed in Table 13.

User Content:

Please split the following solution steps into individual steps and return them formatted as a Python list.
Each step should be a separate string within the list.
If the solution steps contain only a single step or sentence, DO NOT split it further, return it as a single element in the list.
Only return the list itself, with no additional text or formatting.
DO NOT modify the text in any way, simply split it.

Example Format:

```
[
    ”First step description”,
    ”Second step description”,
    ”Third step description”,
    ...
]
```

Solution steps to split: solution

Table 13. Prompt for generating solutions.

F.4. Prompts for Process Evaluation

As described in Section 6, we use the same evaluation for all models to ensure consistency. The prompts for process evaluation are displayed in Table 14.

---

1. System Prompt

You are a teacher skilled in evaluating the intermediate steps of a student's solution to a given problem.

2. User Content

You are given a scientific problem, its correct final answer, and a student's solution to evaluate.
Your task is to: first, solve the problem yourself, using the correct final answer as a hint. Ensure your reasoning leads to the correct answer. Once you have a clear understanding of how the problem could be solved, evaluate the correctness of each step in the student's solution.
Focus exclusively on the scientific, logical, or mathematical correctness of the solution. Ignore differences in formatting, expression style, specific wording, or presentation order, as long as the reasoning and results are valid.

For each step, perform:
1. Binary scoring: assign a score of 1 for correct steps and 0 for incorrect steps.
2. Error classification (only if the step is incorrect):
a. Numerical Calculation Error. Errors in basic arithmetic operations such as addition, subtraction, division, or square roots.
b. Symbolic Calculation Error. Errors in manipulating algebraic expressions, such as incorrect expansion, factoring, simplification, or solving equations with variables.
c. Visual Interpretation Error. Errors in interpreting graphical data, such as misidentifying coordinates, shapes, spatial relationships, or data within figures.
d. Reasoning Error. Errors in the logical thinking process that lead to incorrect conclusions, such as flawed arguments, invalid inferences, or gaps in the logical flow of the solution.
e. Knowledge Error. Errors caused by insufficient understanding or incorrect application of necessary knowledge (e.g., concepts, formulas, theorems, methods), or using outdated or incorrect information.
f. Question Understanding Error. Errors due to misunderstanding or misinterpreting the problems' conditions or requirements, such as misreading questions or misapplying given conditions.
g. No solution provided. The model refuses to answer, fails to follow instructions to make a solution, or encounters anomalies in generation process such as repetitive responses or incomplete outputs.
3. Provide a brief explanation for the identified error.

# The given problem: {problem}

# The Correct Final Answer: {final answer}

# Student's solution: step-by-step student's solution

Finally, your evaluation results should be a Python list within <evaluation> and < /evaluation> tags, as follows:
<evaluation>
[
    ["The full text of a correct step", 1, "", ""],
    ["The full text of an incorrect step", 0, "Error category", "Brief error descrition"],
]
< /evaluation>
Strictly adhere to the output format, with no additional text or formatting.
Ensure the length of your output list matches the student's solution.

Table 14. Prompt for process evaluation.