

Ill-Defined Math: Benchmarking LLM Reasoning Beyond Well-Defined Problems

Huaibo Chen^{1*}, Yixiao Lin^{2*}, Zihan Zhao^{3*}, Pengcheng Chen⁴, Nuohao Liu⁵,
Yue Hu⁶, Qian Xie², Qinbo Bai⁷, Ning Yan⁸, Masood S. Mortazavi⁸, Kamal Youcef-Toumi¹

¹Massachusetts Institute of Technology, ²Cornell University, ³University of California San Diego

⁴University of Washington, ⁵University of Wisconsin-Madison

⁶University of Southern California, ⁷Purdue University, ⁸Futurewei Technologies

Abstract

Ill-defined problems (IDPs) are pervasive in science and engineering, where solutions may fail to exist, be non-unique, or lack a meaningful interpretation. Despite recent advances in large language models (LLMs) in mathematical reasoning, their ability to reason about IDPs remains poorly understood. To address this gap, we introduce Ill-Defined Math (IDM), a benchmark of 1,300 mathematically ill-defined problems whose ill-definedness stems from fundamental mathematical flaws. IDM consists of a test set of 300 expert-constructed and verified problems, along with a larger training set generated via a verified multi-agent pipeline. We further propose a fine-grained, three-stage LLM judge that evaluates responses from three complementary perspectives: final answer statements, issue recognition, and issue fixing. Evaluations on recent LLMs expose clear performance gaps between well-defined and ill-defined problems. Moreover, models frequently recognize ill-definedness during reasoning yet still produce definitive final answers, revealing a systematic disconnect between issue recognition and faithful response behavior. Project page: <https://idmath.github.io>

1 Introduction

Ill-defined problems (IDPs) are ubiquitous across mathematics, science, and engineering. From inverse problems in medical imaging to under-determined systems in mechanics and economics, whether a problem is well-defined fundamentally determines whether a solution exists, is unique, or admits meaningful interpretations. Compared with well-defined problems, reasoning about ill-defined ones often requires additional abilities such as detecting contradictions, ambiguity, or missing conditions, placing qualitatively different demands on a model than standard well-defined problem solving.

Most reasoning benchmarks assume that every problem is well-defined and admits a valid solution path. Yet robust reasoning also requires the ability to recognize when a problem is under-specified, internally inconsistent, or unsolvable as stated. A system that cannot recognize ambiguity, contradiction, or missing conditions may still perform well on conventional benchmarks, yet fail in precisely the situations where reliable reasoning matters most. In this sense, reasoning about ill-defined problems is a core component of robust and trustworthy intelligence.

This regime poses several challenges for large language models (LLMs). First, most math training data consists of well-defined problems Toshniwal et al. (2024); Du et al. (2025), making IDPs underrepresented in typical math training data. Second, reinforcement learning from human feedback (RLHF) may introduce sycophancy Sharma et al. (2024), discouraging models from questioning problem premises. Third, when solutions do not exist, models often generate long chains of thought (CoTs), which may increase opportunities

*Equal contribution. Contact: hbchen@mit.edu, y12883@cornell.edu, ziz078@ucsd.edu

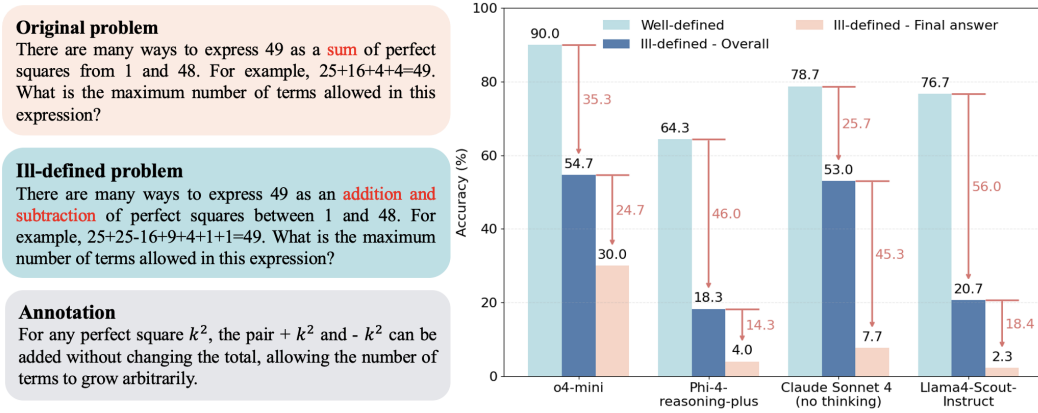


Figure 1: **Left:** Example of an ill-defined problem created by modifying ‘sum’ to ‘addition and subtraction’, leading to an unbounded number of terms. **Right:** Performance comparison of representative LLMs in terms of accuracy on well-defined problems and overall and final-answer (refusal) accuracy on ill-defined problems in the IDM benchmark. Overall accuracy, where the response clearly states the ill-definedness in the final answer or fixes the issues in the middle steps, is substantially lower than the accuracy on well-defined problems. Moreover, accuracy measured by explicit statements of ill-posedness in final answers substantially lower than overall accuracy.

for compounding errors and unsupported reasoning. Despite strong performance on well-posed mathematics Huang & Yang (2025); OpenAI (2025); Guo et al. (2025), IDPs remain an underexplored challenge for current LLMs.

Systematic study of these challenges requires both suitable benchmarks and evaluation protocols, yet existing work falls short on both. Current benchmarks for ill-defined or unsolvable problems mainly focus on superficial flaws or wording ambiguity rather than deep mathematical ill-definedness Zhao et al. (2024); Tian et al. (2024); Ma et al. (2025); Sun et al. (2024); Song et al. (2025); Kirichenko et al. (2025); Fan et al. (2025). Evaluation protocols are often rigid, relying on string matching or single binary judgments Xue et al. (2025); Song et al. (2025); Sun et al. (2024); Ma et al. (2025); Zhao et al. (2024); Kirichenko et al. (2025). When superficial benchmarks are paired with binary evaluation, the resulting metric largely reduces to whether the model outputs a refusal keyword: superficial flaws are easy to detect, so the main discriminator becomes whether the model explicitly says “unsolvable” rather than how deeply it reasons about the underlying issue. Meanwhile, other logically sound behaviors, such as identifying the flaw and proposing a well-founded fix, are penalized under such schemes. Table 1 summarizes these limitations; a detailed overview of related work is deferred to Section 5. To evaluate reasoning over IDPs rather than refusal, two conditions are therefore essential: (1) challenging problems whose ill-definedness requires nontrivial reasoning; and (2) evaluation protocols that recognize diverse correct behaviors, including well-founded fixes.

To simultaneously satisfy these two conditions, we construct Ill-Defined Math (IDM), a benchmark of 1,300 mathematically ill-defined problems, each derived by modifying its well-defined counterpart. An example is shown in the left part of Figure 1. The test set contains 300 problems, each crafted by a PhD-level expert and independently verified by another, with the goal of introducing deep, inherent mathematical ill-definedness. Building on this hand-crafted dataset, we develop a multi-agent pipeline to generate 1,000 problems for use as training data. Unlike prior work Xue et al. (2025); Song et al. (2025); Fan et al. (2025), our pipeline directly generates ill-defined problems from well-defined ones using few-shot examples, employs a verification agent to filter out problems that remain solvable and further removes problems that do not confuse a strong model (GPT-4o OpenAI (2024)). This design is intended to address a common failure mode in previous benchmarks, where automatically generated problems are often still well-defined, and helps ensure that ill-definedness more often arises from intrinsic mathematical flaws rather than superfi-

cial issues. Compared to existing datasets, IDM is designed to provide greater difficulty, stronger quality control, and scalability. Each problem is additionally annotated with a short explanation, summarizing its ill-definedness.

Evaluating responses to IDPs poses its own challenges, as correct behavior may take diverse forms, including explicitly stating ill-definedness, identifying missing or contradictory conditions, or proposing well-founded fixes. Instead of using binary judges Ma et al. (2025); Zhao et al. (2024); Kirichenko et al. (2025), we propose a fine-grained, three-stage LLM-as-judge framework that separately evaluates (i) the final answer statement, (ii) recognition of ill-definedness, and (iii) attempts to fix or reformulate the problem. Responses that are judged correct under either (i) or (iii) are considered correct. This design distinguishes reasoning capability from abstention behavior in the final answer, and reveals failure modes that prior evaluation schemes tend to conflate.

Using IDM and our judge framework, we evaluate 27 state-of-the-art LLMs. Representative results can be found in the right part of Figure 1. Our results reveal a substantial performance gap between well-defined and ill-defined problems, even for the strongest models. We further observe a large gap between accuracy measured by explicit abstention in final answers and overall accuracy, indicating that refusal-based metrics fail to fully capture reasoning capability on ill-defined problems.

In summary, we make four key contributions: (1) IDM, a large-scale benchmark with expert-verified test problems and a high-quality training corpus; (2) a fine-grained, three-stage LLM judge that evaluates reasoning beyond simple refusal; (3) a multi-agent framework with verification and filtering for reliable generation of ill-defined mathematical problems; and (4) a systematic evaluation of 27 recent LLMs that provides new empirical insights into failures under ill-definedness. Together, these contributions provide a benchmark and evaluation framework for studying LLM reasoning on ill-defined problems.

Table 1: Comparison with previous work on ill-defined problems. IDM features an expert-curated and rigorously verified test set with deep, mathematically grounded ill-definedness, together with a scalable training set constructed via a multi-agent generation pipeline with explicit verification. Unlike prior benchmarks that focus on shallow inconsistencies or rely on rigid refusal-based evaluation, IDM pairs the dataset with a multi-stage LLM judge, enabling systematic analysis of diverse reasoning behaviors. Abbreviations: MathTrap Zhao et al. (2024), UMWP Sun et al. (2024), SUM Song et al. (2025), PMC Tian et al. (2024), UMP Ma et al. (2025), ReliableMath Xue et al. (2025), AbstentionBench (Ab.Bench in the table) Kirichenko et al. (2025), MiP Fan et al. (2025), and RoRBench Yan et al. (2025).

Datasets	Source Data Difficulty	Curation Methods		Flaw Type	Evaluation
		Train	Test		
UMWP	Low	✗	Hand crafted	Superficial	Text similarity
SUM	Hard	Auto w/o verif.	✗	Superficial	String match
ReliableMath	Hard	✗	Auto w/o verif.	Still well-defined	String match
MathTrap	Medium	Hand crafted	Hand crafted	Mostly superficial	Single judge
UMP	Low	✗	Auto w/ verif.	Superficial	Single judge
Ab.Bench	Low	✗	Collect & auto	Superficial	Single judge
RoRBench	Low	Auto w/ verif.	Auto w/ verif.	Superficial	Single judge
PMC	Low	✗	Auto w/o verif.	Superficial	Formal lang.
MiP	Low	✗	Auto w/ verif.	Superficial	Two-stage judge
IDM (Ours)	Hard	Auto w/ verif	Hand crafted	Deep, mathematical	Three-stage judge

2 The Ill-Defined Math Dataset

2.1 Task Formalization

Let $\mathcal{D}$ be the domain of unknowns (e.g., $\mathcal{D} = \mathbb{R}^n$ or $\mathcal{D} = \mathbb{Z}^n$). A problem is represented by a finite set of constraints $\mathcal{C} = \{C_1, \dots, C_m\}$ where each C_j is a predicate on $\mathcal{D}$. Define the solution set $S(\mathcal{C}) = \{x \in \mathcal{D} : C_1(x) \wedge \dots \wedge C_m(x)\}$.

Our notion of ill-definedness is adapted from the classical concept of *ill-posedness* in the sense of Hadamard (1902; 1923), which holds that for a problem to be well-posed it should satisfy (i) existence of a solution, (ii) uniqueness of the solution, and (iii) continuous dependence on the data. When these criteria fail, a problem is considered ill-posed. In this work, we focus on existence and uniqueness as they directly affect LLM reasoning and evaluation; stability is outside the scope of our benchmark.

Definition 2.1 (Ill-defined). A problem is *ill-defined* if the stated constraints are either

- **contradictory (infeasible):** the solution set is empty, i.e., $S(\mathcal{C}) = \emptyset$. Or
- **missing-condition:** the solution set contains uncountably many solutions under the intended interpretation (i.e., a continuum), indicating that the constraints are insufficient to determine a unique solution or a countable solution set.

For the missing-condition type, we restrict to cases where the solution set contains a continuum (i.e., uncountably many solutions); problems with finitely or countably many solutions are treated as well-defined. This operational distinction is detailed in Appendix F.

2.2 Dataset Overview

In this subsection, we describe the curation and statistics of Ill-Defined Math (IDM), a novel collection of ill-defined math problems. IDM consists of 1,300 mathematically ill-defined problems, whose ill-definedness arises from deep, fundamental flaws. To ensure reliable, rigorous benchmarking as well as scalability, we separate the dataset into a manually curated and verified test set and an automatically generated training set.

Test data curation. The test set contains 300 problems that were manually modified from original, well-defined problems. The original problems are from diverse sources, including MATH, AIME, AMC, as well as original problems created by the authors. To improve discriminative power, we also adapted 37 questions from MathTrap Zhao et al. (2024). We consider only well-stated problems with unambiguous wording before modification, excluding instances whose sole issue is vagueness or poor phrasing. Modified problems are transformed so that the resultant question meets Definition 2.1. The modification was performed by PhD researchers and Math Olympiad participants. Each problem is also independently checked by another member of our group to ensure quality. We include only problems that Hunyuan-T1 Tencent Hunyuan Team (2025) fails to solve, ensuring sufficient challenge. Manual adaptation is critical because it ensures that contradictory constraints and missing conditions are genuinely present, rather than artifacts of superficial text perturbations. This high-quality test set serves as a reliable and accurate benchmark to evaluate whether models can truly recognize and reason about ill-definedness.

Training set curation. We developed an automated generation pipeline for the training set, designed to scale while preserving the deep, rigorous ill-definedness.

Naively generating ill-defined problems, however, is prone to failure—it often yields instances that remain well-defined or introduce only superficial artifacts rather than deep mathematical issues. To address these challenges, we develop a multi-agent automatic generation pipeline with verification and filtering stages. The details are described in Section 2.3.

Using this pipeline, we construct a training set of 1,000 ill-defined problems derived from well-defined source questions from Nemotron-Math-v2 Du et al. (2025), which has already filtered out simple problems, helping ensure that the resulting problems are sufficiently difficult. Each training instance is accompanied by a short annotation describing its ill-definedness. Together, the training and test sets support both scalable model training and systematic evaluation.

Comparison with existing datasets. Compared with nine prior benchmarks (see Table 1), Ill-Defined Math stands out for (1) providing an expert-curated and rigorously verified

test set whose ill-definedness arises from deep mathematical flaws rather than superficial issues, (2) constructing a training set via a scalable multi-agent generation pipeline with explicit verification paired with annotation, and (3) a fine-grained, multi-stage LLM-as-judge framework that enables the systematic evaluation of diverse model behaviors beyond simple abstention.

2.3 Multi-Agent Pipeline for Problem Generation

We design a multi-agent pipeline that systematically generates, verifies, and filters ill-defined instances at scale, addressing the common failure modes of naive modification (producing still-solvable problems) and superficial perturbation (introducing ambiguity rather than mathematical flaws). A schematic is shown in Figure 2.

At a high level, the pipeline comprises three classes of agents: (i) a *generation agent* that produces candidate ill-defined problems from well-defined originals, (ii) a *verification agent* that checks whether generated candidates meet the formal ill-definedness criteria established in Section 2.1, and (iii) a *filtering block* that ensures the resulting instances are both challenging and non-trivial for strong LLMs.

Generation agent Given a well-defined source problem and a set of hand-crafted few-shot exemplars, the generation agent, powered by Grok-4 xAI (2025), directly produces candidate ill-defined variants and their annotations explaining why they are ill-defined. We provide 20 few-shot exemplars: 10 of the contradictory type and 10 of the missing-condition type. Each exemplar includes the well-defined problem, the ill-defined problem, and a brief rationale for why it is ill-defined. A truncated version of the prompt, containing only 2 exemplars, is shown in Appendix G. If after three trials for the same well-defined problem no candidate passes the verification agent or the filtering block, we skip that question.

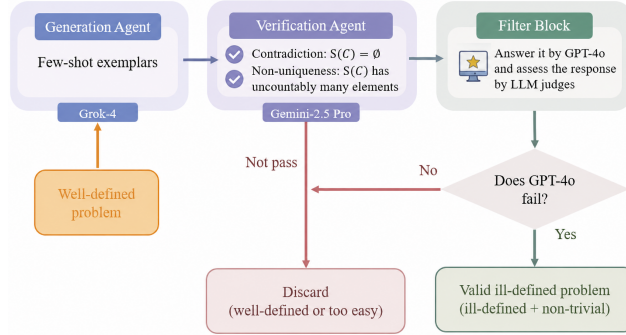


Figure 2: Multi-agent pipeline for generating ill-defined problems. A generation agent (Grok-4) produces candidate ill-defined variants from well-defined sources using few-shot exemplars. A verification agent (Gemini 2.5 Pro) checks whether each candidate satisfies the formal ill-definedness criteria. A filtering block then retains only problems that cause GPT-4o to fail, ensuring the resulting instances are both genuinely ill-defined and non-trivial.

Verification agent Each candidate and its annotation about ill-definedness are passed to a verification agent powered by Gemini 2.5 Pro Google DeepMind (2025), which determines whether the candidate satisfies the ill-definedness criteria. The prompt can be found in Appendix H. Specifically, the agent checks that the constraint set either yields an empty solution set (contradiction) or a continuum of solutions (missing condition), per the formal definitions in Section 2.1. Candidates failing verification are discarded.

Filtering block To further guarantee challenge and eliminate trivial cases, we evaluate each verified candidate using GPT-4o OpenAI (2024). The resulting response from GPT-4o is evaluated using the multi-stage LLM judge, which is described in detail in Section 3.1. Only those instances that cause GPT-4o to produce incorrect responses are retained. This step helps ensure that the benchmark contains problems that are not only formally ill-defined but also non-trivial for a strong contemporary LLM.

The overall pipeline is designed to mitigate several common failure modes of prior automatic generation approaches, producing a large set of mathematically rigorous ill-defined problems suitable for modern LLMs.

3 Evaluation Methods

To evaluate accuracy on the original problems, we directly compare the answers generated by LLMs with the ground-truth answers. We first use the symbolic comparison library adopted from Shao et al. (2024). To handle answers that cannot be recognized by symbolic comparison, those judged incorrect by symbolic comparison are then fed into an LLM to assess equivalence. We use Kimi K2 for this step; the prompt is shown in Appendix I.

3.1 Three-stage LLM Judge

Correct behavior under ill-definedness may take diverse forms, from explicitly stating insolvability to proposing principled modifications. Rigid evaluation schemes such as refusal-keyword matching or binary judgments fail to capture these nuanced behaviors.

To address this challenge, we propose a fine-grained, multi-stage LLM-as-judge framework that decomposes model responses into complementary dimensions of correctness. All stages are powered by GPT-5.6 Luna OpenAI (2026), and the prompts can be found in Appendix J. Rather than conflating reasoning ability with refusal behavior, our framework separately evaluates what a model *states*, what it *recognizes*, and how it *handles* ill-definedness:

Stage I: Final-Answer Statement Judge The final-answer statement judge evaluates whether the model’s final response explicitly asserts that the problem is ill-defined, unsolvable, or cannot be uniquely determined. This stage captures overt refusal or abstention behavior and corresponds most closely to evaluation schemes used in prior benchmarks Kirichenko et al. (2025); Yan et al. (2025); Ma et al. (2025). Importantly, the final statement must match the ill-posedness of the problem, so a naive, overly conservative refusal at this stage is not counted as correct. Conversely, some answers, such as “the maximum does not exist but the upper bound is b ,” are valid final answers in our judge framework, but may not be considered refusal in previous work. The question, the ill-posedness annotation, and the response are provided to the judge.

Stage II: Issue Fixing Judge The issue fixing judge evaluates whether the model attempts to repair the ill-definedness in a principled manner. Valid fixes include introducing explicit assumptions, constraining free variables, or reformulating the problem so that the solution becomes well-defined under clearly stated conditions. Responses that correctly fix the ill-definedness and reason consistently from the modified problem are credited as correct. This design choice captures a realistic scenario that is often not explicitly modeled in prior work on ill-defined math problems. The model does not exhibit refusal or abstention behavior, but “fixing and answering” is logically sound and reasonable.

Stage III: Issue Recognition Judge The issue recognition judge assesses whether the model correctly identifies the source of ill-definedness, such as contradictory constraints or missing conditions, regardless of whether this recognition appears in the final answer or intermediate reasoning. This stage captures a model’s ability to recognize issues during the reasoning steps. For example, a model that identifies a missing condition but nonetheless proceeds to output a definitive answer without fixing the issue would be credited at this stage but penalized elsewhere. This separation enables us to distinguish a model’s realization of anomalies from its ability to correctly handle the issues (explicitly stating the issues in the final answer or fixing them).

The judges are invoked sequentially: stage I first examines the last 100 characters of the response. If it returns true, the process stops, and for reporting purposes issue recognition is treated as true, since success at this stage presupposes recognition of the ill-definedness. Otherwise, stage II is called. If stage II returns true, the process stops again, and for reporting

purposes issue recognition is treated as true, since this outcome also presupposes recognition of the ill-definedness. If stage II returns false, stage III is then called. We define overall correctness as success in either stage I or stage II, since both represent reasonable ways of responding to ill-definedness. Stage III is reported separately as a diagnostic metric to quantify whether the model realizes the issue, even if it ultimately cannot make reasonable responses.

We tune prompts on 84 responses from four representative models, and then validate them on 96 responses from 23 other models. We evaluated the three judges (final-answer, midway-identification, and problem-modification) with human annotations as ground truth on the 96 responses from 24 other models mentioned above. Their performance is summarized in Table 2. All three judges demonstrate strong agreement with human annotations, achieving F1 scores of 0.94 or higher, with near perfect recall. The precision for issue fixing is slightly lower, which is expected given that these behavioral categories are manually defined and issue fixing often involves subtle or borderline cases. Nevertheless, perfect agreement between LLM judges and human annotators is usually hard Sheng et al. (2025), and the resulting precision remains sufficiently high to support evaluation. We also present the confusion matrix for each metric, along with the performance of other two judge models with the same prompts, in Appendix B.1. Although the prompts are not specifically tuned for the other two models, they still achieve reasonably good performance.

Table 2: Consistency between LLM judge and human annotation on 96 test responses.

Model	Acc.	Prec.	Recall	F1-score
Final Answer	1.00	1.00	1.00	1.00
Issue Fixing	0.93	0.89	1.00	0.94
Issue Recognition	0.94	0.94	0.98	0.96
Overall Assessment	0.95	0.92	1.00	0.96

4 Experimental Results

4.1 Performance of LLMs

We evaluated 27 modern LLMs on IDM in a zero-shot setting using the following instruction: “You are a math expert. Please reason step by step and show your step-by-step reasoning first, and then put your final answer within `\boxed{}`”, using the generation parameters in Table C1. On non-empty responses, we report three metrics in Table 3: final answer accuracy (insolvability declared in the final response), identification accuracy (insolvability flagged during reasoning), and overall accuracy (problem recognized as insolvable if stage I or stage II succeeds).

From the benchmarking results, we find that ill-defined problems substantially degrade performance across nearly all evaluated LLMs. Accuracy on ill-defined instances is consistently lower than on well-defined ones, even for the strongest models. For some models, such as GPT-4o and phi-4-reasoning-plus, performance degradation can reach around 40%. A small number of models show near-zero or slightly negative accuracy drops, which we attribute to two factors: the Issue Fixing judge’s tendency toward false positives (Figure B3), which inflates overall accuracy on IDPs, and variance from the moderate test set size of 300 problems, where a few percentage points fall within the noise floor. Even for these models, final answer accuracy remains far below well-defined accuracy, confirming that the abstention gap persists regardless. Since the ill-defined counterparts require similar knowledge and skills to the original ones, these gaps indicate that ill-definedness reveals failure modes that standard mathematical benchmarks may not fully capture, and that progress on well-posed problems does not directly translate to robustness under ill-defined conditions.

Beyond aggregate accuracy drops, the fine-grained metrics reveal two systematic behavioral gaps that characterize LLM reasoning under ill-definedness. First, we observe a large Abstention Gap: the average overall correctness on ill-defined problems is 61.7%, whereas the average final-answer accuracy is only 14.5%, yielding a gap of 47.3%. This indicates that models frequently generate reasonable responses by correcting the issues, such as introducing new assumptions or modifying constraints, yet they fail to explicitly state that

Model	Ill-defined problem acc			Well-defined Acc	Acc Drop	Handling Gap	Abstention Gap
	Final Answer	Recog.	Overall				
Open Chat Models							
DeepSeek-V3.1	10.0	82.7	77.7	84.7	7.0	5.0	67.7
Kimi-K2-Instruct-0905	20.7	71.7	61.7	82.0	20.3	10.0	41.0
Llama3.3-70B-Instruct	0.7	32.0	13.3	67.3	54.0	18.7	12.7
Llama4-Maverick-Instruct	4.0	51.7	30.3	77.0	46.7	21.3	26.3
Llama4-Scout-Instruct	2.3	41.3	20.7	76.7	56.0	20.7	18.3
Qwen2.5-72B-Instruct	3.3	36.3	19.3	67.7	48.4	17.0	16.0
Qwen3-235B-A22B-Instruct-2507	9.7	93.0	88.3	88.0	-0.3	4.7	78.7
Qwen3-30B-A3B-Instruct-2507	8.0	90.0	86.0	86.3	0.3	4.0	78.0
Open-source Reasoning LLMs							
DeepSeek-V3.1-Think	11.0	95.7	89.7	94.7	5.0	6.0	78.7
GPT-oss-120b	27.3	91.7	87.3	87.7	0.4	4.3	60.0
GPT-oss-20b	30.7	90.0	84.7	87.3	2.6	5.3	54.0
Phi-4-reasoning-plus	4.0	34.3	18.3	64.3	46.0	16.0	14.3
Qwen3-235B-A22B-Thinking-2507	7.3	94.0	71.0	91.0	20.0	23.0	63.7
Qwen3-30B-A3B-Thinking-2507	8.3	94.3	84.0	92.0	8.0	10.3	75.7
Qwen3-8B (Thinking)	5.0	92.3	82.3	80.7	-1.6	10.0	77.3
Qwen2.5-Math-72B-Instruct	5.3	36.0	22.0	70.0	48.0	14.0	16.7
Proprietary Chat LLMs							
Gemini 2.5 Flash (no thinking)	22.7	88.7	81.0	89.3	8.3	7.7	58.3
Claude Sonnet 4 (no thinking)	7.7	65.7	53.0	78.7	25.7	12.7	45.3
GPT-4o-2024-11-20	18.0	30.0	22.3	61.7	39.4	7.7	4.3
GPT-4.1-2025-04-14	25.7	73.0	59.7	76.7	17.0	13.3	34.0
Proprietary Reasoning LLMs							
Claude Sonnet 4 (extended thinking)	13.3	86.0	76.0	86.3	10.3	10.0	62.7
Gemini 2.5 Flash (with thinking)	26.0	91.3	83.0	87.3	4.3	8.3	57.0
Gemini 2.5 Pro	19.0	90.7	79.7	91.3	11.6	11.0	60.7
Grok-4 ¹	19.5	64.9	58.4	95.9	37.5	6.5	39.0
Grok-3-mini-beta	5.3	92.3	81.7	88.7	7.0	10.7	76.3
o4-mini-2025-04-16	30.0	58.7	54.7	90.0	35.3	4.0	24.7
GPT-5-2025-08-07	45.5	84.6	80.6	90.7	10.1	4.0	35.1
Average Acc	14.5	72.3	61.7	82.7	21.0	10.6	47.3

Table 3: Evaluation performance of 27 LLMs on the Ill-Defined Math (IDM) benchmark test set. We report accuracy on well-defined problems (Well-defined Acc) and three metrics for ill-defined problems: Final Answer Accuracy, Issue Recognition (Recog.) Accuracy, and Overall Accuracy. We further report the Handling Gap (Recog. Acc. - Overall Acc.), which measures failures to properly handle recognized issues, and the Abstention Gap (Overall Acc. - Final Answer Acc.), which measures the discrepancy between correct reasoning and the explicit final-answer claim. Underlining denotes the highest value within each model category, while boldface highlights the highest value across all models.

the problems are ill-defined in their final answers. Second, we identify a Handling Gap between issue recognition and overall correctness (10.6% on average). This gap shows that recognizing ill-definedness does not guarantee correct handling: many models correctly identify missing or contradictory conditions but nevertheless proceed to produce definitive answers without resolving the underlying issues.

Finally, performance patterns differ systematically across model families and reasoning modes. Reasoning models generally achieve higher recognition and overall accuracies than their chat counterparts, confirming that reasoning capability (although generally measured by well-defined problems) helps identify and handle ill-definedness. However, these same models often exhibit large Handling and Abstention Gaps, suggesting that improved reasoning alone is insufficient for faithful behavior under ill-definedness.

The evaluation should be interpreted with several caveats. First, patterns such as issue recognition and issue fixing do not always admit strict boundaries, and some cases remain inherently ambiguous even to human annotators. Second, LLM judges often struggle to recognize subtle differences between these patterns, making it difficult to simultaneously reduce false positives and false negatives through prompt tuning; our test-set analysis suggests that the reported accuracy on ill-defined problems is somewhat overestimated. Third, for many proprietary models, the full chain-of-thought is unavailable, so issue recognition or fixing may occur internally without being reflected in the observable response.

¹For Grok-4, there are 69 responses with empty outputs among the 300 problems. This likely indicates that the chain-of-thought generation exhausted the 20,000-token budget, leaving no remaining tokens for the final answer. Consequently, the reported results for Grok-4 may be substantially biased.

Therefore, the benchmark results are best viewed as a qualitative characterization of model behavior and an estimate of broad capability ranges, rather than a precise ranking of model accuracies.

4.2 Failure Mode Analysis

We identify three common failure modes, illustrated with examples in Figure E1. The corresponding annotation for ill-definedness is in Figure D1.

(1) Guessing. In this case, the model shows signs of detecting contradictions or anomalies and then attempts different methods or values. After realizing none of them work, it nevertheless produces a definitive final answer without justification (see Figure E1, Case 1). This behavior likely reflects an inherent bias toward providing a definite answer rather than admitting impossibility.

(2) Hallucination. In this case, the model produces intermediate steps containing spurious claims that patch missing conditions or reconcile contradictions. Typical patterns include fabricated identities, arbitrary substitutions, or even basic arithmetic errors, leading to a meaningless final answer (see Figure E1 Case 2).

(3) Condition Neglect. In this failure mode, the model follows an apparently rigorous reasoning path but overlooks an implicit condition, producing an answer that violates it (see Figure E1, Case 3). This pattern often arises in problems where even humans must carefully verify solutions to avoid mistakes, such as checking domain requirements for periodic functions in Case 3 of Figure E1.

Across the three failure modes, models also exhibit two recurring fallback behaviors when confronted with difficult instances. First, they often resort to brute-force enumeration or exhaustive case checking rather than delivering a concise analytical diagnosis. Second, they expand reasoning into long chains of low-level arithmetic computations, which inflate verbosity and increase the risk of simple mistakes. A similar phenomenon is also observed in Fan et al. (2025).

4.3 Discussion

The results show that performance on ill-defined problems drops substantially compared with well-defined counterparts, and that LLMs often produce definitive answers even when they detect ill-definedness. Based on our observed patterns, we hypothesize that these failures originate from two factors.

One possible factor is that ill-defined problems are underrepresented in typical math training data. Most math training corpora consist primarily of well-defined problems with definitive final answers, which may bias models toward treating answer production as the default objective of reasoning. As a result, under ill-defined settings, models may recognize anomalies during reasoning but still output a concrete answer by implicitly assuming additional constraints or hallucinating, rather than questioning the problem’s premises.

Another plausible explanation relates to sycophantic tendencies introduced during RLHF. Prior work suggests that RLHF encourages models to trade off truthfulness for user agreement Sharma et al. (2024). In ill-defined settings, this may discourage explicit rejection or premise questioning, leading models to guess even when no valid solution can be found, consistent with our observed failure patterns.

5 Related Work

5.1 Benchmarks for Math Reasoning

To evaluate LLMs’ reasoning capability, numerous math reasoning benchmarks have been proposed. Representative ones include GSM8K, MATH, Minerva, OlympiadBench, AMC, and AIME Cobbe et al. (2021); Hendrycks et al. (2021); Lewkowycz et al. (2022); He et al. (2024). As LLM reasoning improves, the community increasingly calls for higher-difficulty

math benchmarks to avoid saturation. Indeed, some recently released models, such as Grok-4 and GPT-5, already exceed 90% accuracy on AIME.

5.2 Variants of Math Benchmarks

In addition to proposing more challenging math benchmarks, another line of work focuses on constructing variants of existing benchmarks to assess data contamination, memorization, and robustness. Common approaches include introducing irrelevant information Shi et al. (2023); Rajeev et al. (2025), converting static benchmarks into dynamic or functional formats Srivastava et al. (2024); Mirzadeh et al. (2024), mutating existing problems to create more challenging variants Shah et al. (2024), and applying minimal edits to test memorization, where even small perturbations can substantially alter the underlying logic Huang et al. (2025).

5.3 Benchmarks of Insolvable Math Problems

Research on insoluble or unanswerable math problems constitutes another closely related direction. We compare IDM with prior benchmarks in Table 1. MathTrap Zhao et al. (2024), UMP Ma et al. (2025), and UMWP Sun et al. (2024) are pioneering efforts in this area, but their datasets are primarily derived from elementary-level problems and often center on commonsense inconsistencies or ambiguous wording, with limited mathematical depth. In addition, their evaluation relies on a binary, single LLM-based judge. PMC Tian et al. (2024) scales problem generation by LLMs, but many are relatively simple, and automatic transformations can introduce only superficial forms of insolubility or solvable cases. ReliableMath Xue et al. (2025) extends this direction to competition-level mathematics, but its automatic generation pipeline can still produce “pseudo-insoluble” instances, while its string-matching evaluation may underestimate model performance. RoRBench Yan et al. (2025) studies recitation versus reasoning by introducing subtle perturbations to relatively simple problems, with many perturbations focusing on commonsense changes. MiP Fan et al. (2025) focuses on missing-condition problems, with most instances drawn from elementary mathematics. The math subset of AbstentionBench Kirichenko et al. (2025) is likewise primarily elementary. SUM Song et al. (2025) studies refusal behavior through reinforcement learning, but the forms of insolubility are generally superficial and less mathematical. As discussed above, without fine-grained evaluation and problems exhibiting deep, rigorous forms of ill-definedness, benchmark results may be unreliable or may primarily measure refusal behavior rather than reasoning ability.

6 Conclusion

We introduced Ill-Defined Math (IDM), a benchmark of 1,300 mathematically ill-defined problems with an expert-verified test set and a scalable training set generated via a multi-agent pipeline. We also proposed a multi-stage LLM-as-judge framework that distinguishes issue recognition, issue fixing, and final-answer statement. Our evaluation shows a clear performance gap between ill-defined and well-defined problems, and a discrepancy between sound reasoning and direct claims in final answers. These findings highlight ill-defined problems as a challenging frontier for faithful LLM reasoning.

Acknowledgement

We would like to thank Dr. Pan Lu at Stanford University, Mr. Jiayi Sheng at Sichuan University and Dr. Hejie Cui at Google for helpful discussions. We would also like to thank the paid contributors for providing many creative and high-quality ill-defined problems. The contributors are listed below:

Contributors	Number of Problems	Affiliation
Dingyu Wu	50	Renmin University of China
Yihang Song	47	Beijing University of Posts and Telecommunications
Kefan Cao	35	Peking University
Jiandong Wu	24	Shanghai Yangjing High School
Jingxuan Yan	17	ShanghaiTech University
Ziqi Guo	21	Purdue University
Dingyu Chen	31	Shanghai University
Tiantong Hu	11	Westtown School
Yonghao Zhu	7	Shanghai Jiao Tong University
Wenjie Ji	1	Shanghai University

Reproducibility Statement

We provide full prompt templates for all pipeline stages (generation, verification, equivalence checking, and judge evaluation) in the appendix. The prompt files and scripts for judge and problem generation are also in the GitHub repository (see project page <https://idmath.github.io>). Sampling parameters for all benchmarked models are listed in Appendix C.

Ethics Statement

This work aims to advance the evaluation and understanding of LLM reasoning where problem specifications are incomplete or internally inconsistent. By introducing a benchmark and evaluation framework for ill-defined mathematical problems, our contribution is primarily methodological and diagnostic, with the goal of enabling more reliable and faithful reasoning systems.

We do not anticipate direct negative societal impacts arising from this work. On the contrary, improved evaluation of model performance under ill-defined settings may help identify failure modes that are relevant to real-world applications in science and engineering, where imperfect or ambiguous specifications are common. By highlighting gaps between issue recognition, issue fixing, and final answer behavior, this work may also inform future efforts to improve reasoning robustness in LLMs.

As with most foundational research on model evaluation, the broader societal implications depend on how future models and systems incorporate these insights. We believe that the tools and analyses presented here contribute positively to the responsible development of language models, without introducing new ethical risks beyond those already well recognized in the field.

References

- Rohith Reddy Bellibatlu, Edward Raff, and Wenbin Zhang. Judgesense: A benchmark for prompt sensitivity in llm-as-a-judge systems, 2026. URL <https://arxiv.org/abs/2604.23478>.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Łukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint*, 2021. URL <https://arxiv.org/abs/2110.14168>.
- Wei Du, Shubham Toshniwal, Branislav Kisacanin, Sadegh Mahdavi, Ivan Moshkov, George Armstrong, Stephen Ge, Edgar Minasyan, Feng Chen, and Igor Gitman. Nemotron-math: Efficient long-context distillation of mathematical reasoning from multi-mode supervision, 2025. URL <https://arxiv.org/abs/2512.15489>.
- Chenrui Fan, Ming Li, Lichao Sun, and Tianyi Zhou. Missing premise exacerbates overthinking: Are reasoning models losing critical thinking skill? In *Second Conference on Language Modeling*, 2025. URL <https://openreview.net/forum?id=ufozo2Wc9e>.
- Google DeepMind. Gemini 2.5 Pro. <https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-5-pro>, 2025. Accessed: 2025-10-23.
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shiron Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025.
- Jacques Hadamard. Sur les problèmes aux dérivées partielles et leur signification physique. *Princeton University Bulletin*, 1902.
- Jacques Hadamard. *Lectures on Cauchy’s Problem in Linear Partial Differential Equations*. Yale University Press, 1923.
- 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. In *NeurIPS Datasets and Benchmarks Track*, 2021. URL <https://arxiv.org/abs/2103.03874>.
- Kaixuan Huang, Jiacheng Guo, Zihao Li, Xiang Ji, Jiawei Ge, Wenzhe Li, Yingqing Guo, Tianle Cai, Hui Yuan, Runzhe Wang, Yue Wu, Ming Yin, Shange Tang, Yangsibo Huang, Chi Jin, Xinyun Chen, Chiyuan Zhang, and Mengdi Wang. MATH-Perturb: Benchmarking llms’ math reasoning abilities against hard perturbations. In *Proceedings of the 42nd International Conference on Machine Learning (ICML 2025) – Poster*, 2025. URL <https://icml.cc/virtual/2025/poster/45435>.
- Yichen Huang and Lin F Yang. Winning gold at imo 2025 with a model-agnostic verification-and-refinement pipeline. *arXiv preprint arXiv:2507.15855*, 2025.
- Polina Kirichenko, Mark Ibrahim, Kamalika Chaudhuri, and Samuel J. Bell. Abstention-bench: Reasoning llms fail on unanswerable questions, 2025. URL <https://arxiv.org/abs/2506.09038>.
- Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in neural information processing systems*, 35:3843–3857, 2022.

- Jingyuan Ma, Damai Dai, Zihang Yuan, Rui li, Weilin Luo, Bin Wang, Qun Liu, Lei Sha, and Zhifang Sui. Large language models struggle with unreasonability in math problems, 2025. URL <https://arxiv.org/abs/2403.19346>.
- Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models, 2024. URL <https://arxiv.org/abs/2410.05229>.
- OpenAI. GPT-4o. <https://platform.openai.com/docs/models/gpt-4o>, 2024. Accessed: 2025-10-23.
- OpenAI. GPT-5. <https://platform.openai.com/docs/models/gpt-5>, 2025. Accessed: 2025-10-23.
- OpenAI. gpt-5.6-luna. <https://developers.openai.com/api/docs/models/gpt-5.6-luna>, 2026. Accessed: 2025-08-08.
- Meghana Arakkal Rajeev, Rajkumar Ramamurthy, Prapti Trivedi, Vikas Yadav, Oluwanifemi Bamgbose, Sathwik Tejaswi Madhusudhan, James Zou, and Nazneen Rajani. Cats confuse reasoning LLM: Query agnostic adversarial triggers for reasoning models. In *Second Conference on Language Modeling*, 2025. URL <https://openreview.net/forum?id=VrEPiN5WhM>.
- Vedant Shah, Dingli Yu, Kaifeng Lyu, Simon Park, Jiatong Yu, Yinghui He, Nan Rosemary Ke, Michael Mozer, Yoshua Bengio, Sanjeev Arora, et al. Ai-assisted generation of difficult math questions. *arXiv preprint arXiv:2407.21009*, 2024.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. URL <https://arxiv.org/abs/2402.03300>.
- Mrinank Sharma, Meg Tong, Tomasz Korbak, David Duvenaud, Amanda Askell, Samuel R. Bowman, Esin DURMUS, Zac Hatfield-Dodds, Scott R Johnston, Shauna M Kravec, Timothy Maxwell, Sam McCandlish, Kamal Ndousse, Oliver Rausch, Nicholas Schiefer, Da Yan, Miranda Zhang, and Ethan Perez. Towards understanding sycophancy in language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=tvhaxkMKAn>.
- Jiayi Sheng, Luna Lyu, Jikai Jin, Tony Xia, Alex Gu, James Zou, and Pan Lu. Solving inequality proofs with large language models. *arXiv preprint arXiv:2506.07927*, 2025. URL <https://arxiv.org/abs/2506.07927>. Accepted to NeurIPS 2025 (Spotlight).
- Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H. Chi, Nathanael Schärli, and Denny Zhou. Large language models can be easily distracted by irrelevant context. In *arXiv preprint arXiv:2302.00093*, 2023. URL <https://arxiv.org/abs/2302.00093>.
- Linxin Song, Taiwei Shi, and Jieyu Zhao. The hallucination tax of reinforcement finetuning. *arXiv preprint arXiv:2505.13988*, 2025.
- Sadman Kabir Soumik. Judging the judges: A systematic evaluation of bias mitigation strategies in LLM-as-a-judge pipelines. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL <https://openreview.net/forum?id=QF41AmG4zc>.
- Saurabh Srivastava, Annarose M B, Anto P V, Shashank Menon, Ajay Sukumar, Adwaith Samod T, Alan Philipose, Stevin Prince, and Sooraj Thomas. Functional benchmarks for robust evaluation of reasoning performance, and the reasoning gap. *arXiv preprint arXiv:2402.19450*, 2024. URL <https://arxiv.org/abs/2402.19450>.
- YuHong Sun, Zhangyue Yin, Qipeng Guo, Jiawen Wu, Xipeng Qiu, and Hui Zhao. Benchmarking hallucination in large language models based on unanswerable math word problem. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pp. 2178–2188, 2024.

Tencent Hunyuan Team. Reasoning efficiency redefined! meet tencent’s hunyuan-t1—the first mamba-powered ultra-large model. https://tencent.github.io/llm.hunyuan.T1/README_EN.html, 2025. Accessed: 2026-08-08.

Shi-Yu Tian, Zhi Zhou, Kun-Yang Yu, Ming Yang, Lin-Han Jia, Lan-Zhe Guo, and Yu-Feng Li. Vc search: Bridging the gap between well-defined and ill-defined problems in mathematical reasoning, 2024. URL <https://arxiv.org/abs/2406.05055>.

Shubham Toshniwal, Ivan Moshkov, Sean Narenthiran, Daria Gitman, Fei Jia, and Igor Gitman. Openmathinstruct-1: A 1.8 million math instruction tuning dataset. *arXiv preprint arXiv: Arxiv-2402.10176*, 2024.

xAI. Grok-4. <https://x.ai/news/grok-4>, 2025. Accessed: 2025-10-23.

Boyang Xue, Qi Zhu, Rui Wang, Sheng Wang, Hongru Wang, Fei Mi, Yasheng Wang, Lifeng Shang, Qun Liu, and Kam-Fai Wong. Reliablemath: Benchmark of reliable mathematical reasoning on large language models, 2025. URL <https://arxiv.org/abs/2507.03133>.

Kai Yan, Yufei Xu, Zhengyin Du, Xuesong Yao, Zheyu Wang, Xiaowen Guo, and Jiecao Chen. Recitation over reasoning: How cutting-edge language models can fail on elementary school-level reasoning problems? In *ICML Workshop on Reliable and Responsible Foundation Models*, 2025.

Jun Zhao, Jingqi Tong, Yurong Mou, Ming Zhang, Qi Zhang, and Xuanjing Huang. Exploring the compositional deficiency of large language models in mathematical reasoning through trap problems. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 16361–16376, Miami, Florida, USA, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.915. URL <https://aclanthology.org/2024.emnlp-main.915>.

A Disclosure of LLM Usage

This work uses large language models in two central capacities: (1) as part of a multi-agent pipeline for automated generation and verification of ill-defined problems (Section 2.3), and (2) as the three-stage LLM judge for evaluating model responses (Section 3.1). All LLM-generated data were subject to expert human verification. LLMs evaluated in the benchmark serve as test subjects. Minor LLM assistance was used during paper writing for grammar checking.

B Judge Verification

B.1 Confusion Matrix with Human Annotation

The confusion matrix of our judge with human annotation as ground truth is shown in Figure B3. As is shown in the figure, for most samples the prediction of judge aligns with human annotation, with only a few wrong ones, mainly false positive. This indicates that the judge can overestimate the issue fixing, issue recognition, and overall accuracy.

B.2 Sensitivity to Different Judge Models

We also show the accuracy (with human annotation as ground truth) on the 92 test samples of different judge models with the same prompt. As shown in Table B4, replacing the judge model while keeping the evaluation prompt fixed still yields strong agreement with human annotations, with overall F1 scores of 0.915 and 0.959 for Claude Opus 4.8 and Grok 4.5, respectively. Nevertheless, we argue that cross-model consistency is not a prerequisite for a reliable LLM judge, as judging behavior is inherently sensitive to the choice of model. JudgeSense Bellibatlu et al. (2026) reports substantial variation in prompt sensitivity across judge models and finds that model scale does not predict judging consistency, while *Judging the Judges* Soumik (2026) shows that the effectiveness of bias-mitigation strategies is strongly

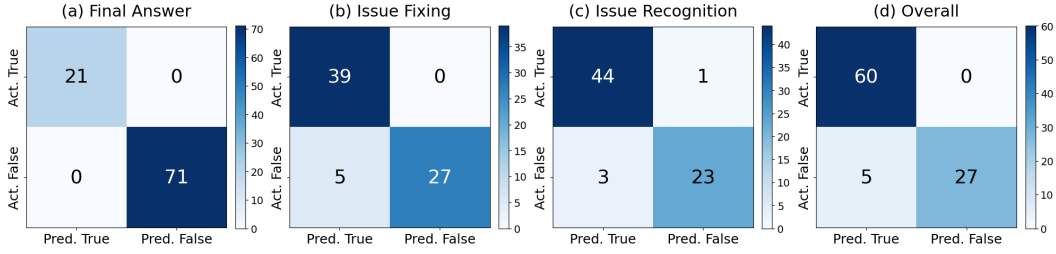


Figure B3: Confusion matrix of 92 test samples.

Table B4: Agreement between human annotations and different LLM judges using the same evaluation prompt.

Judge Model	Criterion	Acc.	Prec.	Rec.	F1
Claude Opus 4.8	Final Answer	0.957	0.905	0.905	0.905
	Issue Recognition	0.930	0.933	0.955	0.944
	Issue Fixing	0.873	0.892	0.868	0.880
	Overall	0.891	0.931	0.900	0.915
Grok 4.5	Final Answer	0.957	0.947	0.857	0.900
	Issue Recognition	0.918	0.913	0.955	0.933
	Issue Fixing	0.918	0.881	0.974	0.925
	Overall	0.946	0.951	0.967	0.959

model-dependent. These findings suggest that the judge model and its prompt should be treated jointly as an evaluator configuration rather than requiring a universal prompt that behaves identically across models. Therefore, it is sufficient to tune the prompt for a fixed judge model on a development set and validate the resulting evaluator on an independent human-annotated test set.

C Sampling Parameters for Benchmarking

The sampling parameters used for each model are shown in Table C1.

D Annotation of Ill-Definedness for Failure Pattern Analysis

See Figure D1.

Table C1: List of LLMs with their default parameters

Model	Size (B)	Temp.	Top_p	Top_k	Max_tokens	Reasoning effort
<i>Open Chat Models</i>						
DeepSeek-V3.1	671	0	N/A	N/A	20000	
Kimi-K2-Instruct-0905	1000	0.6	0.95	N/A	20000	
Llama3.3-70B-Instruct	70	0.6	0.95	20	20000	
Llama4-Maverick-Instruct	402	0	N/A	N/A	20000	
Llama4-Scout-Instruct	109	0	N/A	N/A	20000	
Qwen2.5-72B-Instruct	72	0.6	0.95	20	20000	
Qwen3-235B-A22B-Instruct-2507	238	0.7	0.8	20	20000	
Qwen3-30B-A3B-Instruct-2507	30	0.7	0.8	20	20000	
<i>Open-source Reasoning LLMs</i>						
DeepSeek-V3.1-Think	671	0	N/A	N/A	20000	
GPT-oss-120b	120	1	1	N/A	20000	
GPT-oss-20b	20	1	1	N/A	20000	
Phi-4-reasoning-plus	14.7	0.8	0.95	50	20000	
Qwen3-235B-A22B-Thinking-2507	238	0.6	0.95	20	20000	18000
Qwen3-30B-A3B-Thinking-2507	30	0.6	0.95	20	20000	18000
Qwen3-8B (Thinking)	8	0.6	0.95	20	20000	18000
Qwen2.5-Math-72B-Instruct	72	0	N/A	N/A	4096	
<i>Proprietary Chat LLMs</i>						
Gemini 2.5 Flash (no thinking)	–	1	0.95	64	20000	
claude-sonnet-4-20250514 (no thinking)	–	1	0.95	N/A	20000	
GPT-4o-2024-11-20	–	0.6	0.95	N/A	16384	
GPT-4.1-2025-04-14	–	0.6	0.95	N/A	20000	
<i>Proprietary Reasoning LLMs</i>						
claude-sonnet-4-20250514 (extended thinking)	–	1	0.95	N/A	20000	18000
Gemini 2.5 Flash (with thinking)	–	1	0.95	64	20000	18000
Gemini 2.5 Pro	–	1	0.95	64	20000	18000
grok-4	–	0.6	0.95	N/A	20000	
grok-3-mini-beta	–	0.6	0.95	N/A	20000	high
o4-mini-2025-04-16	–	N/A	N/A	N/A	20000	medium
GPT-5-2025-08-07	–	N/A	N/A	N/A	20000	medium

Insolvable Problem 1: Matrix Determinant	
Insolvable Problem	Annotation
Missing Condition Consider real numbers x, y, z, u, v, w such that the determinant of the matrix with elements (u, y, z) in first row, (x, v, z) in second row, and (x, y, w) in third row is 1. Given that $x \neq u, y \neq v, z \neq w$, determine the value of $u/(u - x) + v/(v - y) + w/(w - z)$.	We are given that the determinant equals $uvw - uyz - vxz - wxy + 2xyz = 1$. We fix $x = 0, y = 0, z = 1$. Then the determinant reduces to $uvw = 1$. The desired sum becomes $u/(u - 0) + v/(v - 0) + w/(w - 1) = u + v + w/(w - 1) = 3 + 1/(w - 1)$. You can choose any w other than 0 and 1 to change the value of the desired formula, and you always can find valid u, v that can satisfy the given constraints.
Insolvable Problem 2: Functional Equations	
Insolvable Problem	Annotation
Missing Condition $h(x)$ and $k(x)$ satisfy $h(k(x)) = x^2$ and $k(h(x)) = x^3$ for all $x \geq 1$. Given that $k(16) = 16$, what is the value of $[k(8)]^3$?	Applying $k()$ to the first condition, and substituting $x = k(x)$ into the second condition, we can obtain $k(x^2) = [k(x)]^3$. From $k(16) = 16 = [k(4)]^3$ we get $k(4) = \sqrt[3]{16}$. Similarly, we get $k(2) = 16^{1/9}$. However, 8 is not a perfect square, so the value $k(8)$ (and hence $[k(8)]^3 = k(64)$) is not determined.
Insolvable Problem 3: Periodic Function	
Insolvable Problem	Annotation
Contradiction The smallest positive period of the function $f(x) = 2 \tan x / (1 - k \tan^2 x)$ is $\pi/2$, find the value of the positive number k.	The domain of f excludes $x = \pi/2 + n\pi, n \in \mathbb{Z}$, where $\tan x$ blows up. Since f is invariant under a shift of $\pi/2$, one must also make $x = n\pi, n \in \mathbb{Z}$ out of domain. However, no matter how we choose k, we cannot make $x = n\pi, n \in \mathbb{Z}$ out of domain.

Figure D1: Failure Pattern Analysis Trap Problem Annotation

E Failure Mode Examples

See Figure E1.

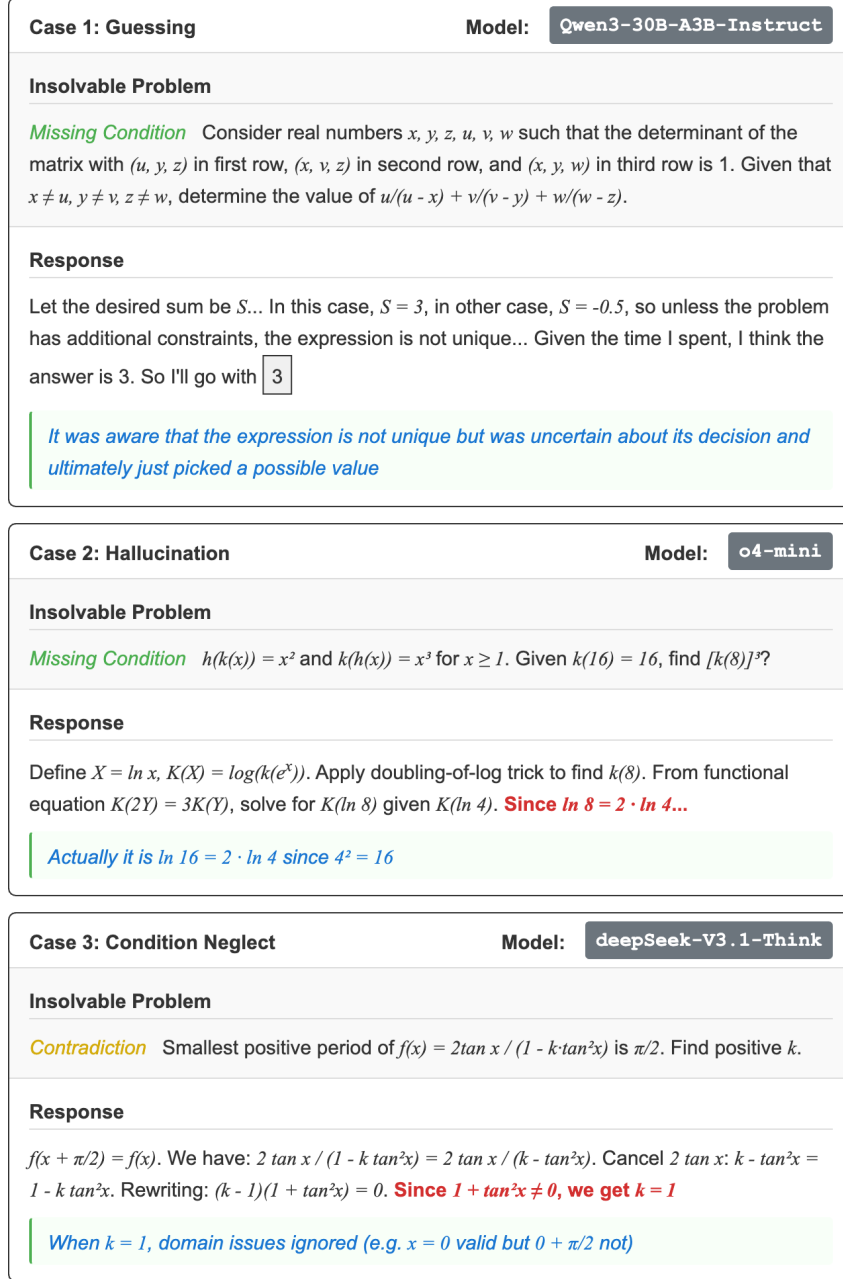


Figure E1: Sample problems for Failure Pattern Analysis. Case 1: Guessing, Case 2: Hallucination, Case 3: Condition Neglect.

F Task Formalization Details

We distinguish underdetermined cases as follows:

1. **Discrete underdetermined:** If $S(\mathcal{C})$ is finite or countably infinite (typical when $\mathcal{D} = \mathbb{Z}^n$), we treat the problem as well-defined in the usual sense, since solutions

can, in principle, be enumerated; such instances are not labeled missing-condition in our dataset.

2. **Continuous underdetermined:** If $S(\mathcal{C})$ contains a continuum (typical when $\mathcal{D} = \mathbb{R}^n$), often parameterized by free variables, then the instance is labeled missing-condition under our intended uniqueness assumption.

This distinction reflects pragmatic considerations for dataset design and evaluation, rather than a fundamental mathematical dichotomy: in general mathematics, other aspects of ill-posedness (e.g., stability) may fail even when existence and uniqueness conditions are satisfied Hadamard (1902; 1923). Instances with a finite or countably infinite set of solutions (e.g., $x = k\pi, k = 1, 2, 3, \dots$) are not labeled as ill-defined in our dataset.

G Prompt for Generation Agent

You are a **Generation Agent** for creating **ill-defined math problems** from well-defined source problems. Your goal is to produce an ill-defined variant via **minimal edits** that introduce an **intrinsic mathematical flaw**, not superficial rewording.

Input:

Source Problem: a well-defined math problem.

Output (a triplet):

- **Source Problem:** repeat the original problem verbatim (or with only formatting normalization).
- **Ill-defined Variant:** a minimally edited version of the source that becomes ill-defined.
- **Annotation:** a short, rigorous explanation of why the variant is ill-defined *under the intended interpretation*.

Ill-definedness types you may target (choose exactly one per output):

- **Contradictory (infeasible):** the constraints force an impossibility so that no solution exists.
- **Missing condition:** the constraints are insufficient so the answer is not uniquely determined, typically yielding **uncountably many** valid solutions (a continuum), or making a required sub-expression undefined.

Hard constraints (must follow):

1. **Minimal edit rule.** Edit budget: make at most **1–3 localized changes** (e.g., change one condition, add one extra constraint, or alter one case in a definition). Keep the topic, symbols, and target question the same.
2. **No superficial perturbations.** Do *not* rewrite for style, add distractor sentences, or change vocabulary substantially. The source and variant should be nearly identical in wording and structure.
3. **Intrinsic flaw rule.** The variant must be ill-defined due to a genuine mathematical issue: contradiction (no solution) or missing condition (continuum of solutions), not due to ambiguity of language.
4. **Annotation must be checkable.** Provide a concise argument/construction witnessing the flaw (derive a contradiction, show a free real parameter, or show non-termination/undefinedness).
5. **Do not fix.** Do not propose repaired assumptions; only generate the ill-defined variant and explain the flaw.

Few-shot exemplar A (Missing condition).

Source Problem:

Define the function

$$q(y, z) = \begin{cases} y + z & \text{if } y \geq 0 \text{ and } z \geq 0, \\ y - 2z & \text{if } y < 0 \text{ and } z < 0, \\ 3y + z & \text{otherwise.} \end{cases}$$

Determine the value of $q(q(1, -1), q(-5, -2))$.

Ill-defined Variant:

Define the function

$$q(y, z) = \begin{cases} y + z & \text{if } y \geq 0 \text{ and } z \geq 0, \\ y - 2z & \text{if } y \geq 0 \text{ and } z < 0, \\ q(y, -2z) & \text{otherwise.} \end{cases}$$

Suppose that $q(-1, 1) = 1$. Determine the value of $q(q(1, -1), q(-5, -2))$.**Annotation:**

$$q(-5, -2) = q(-5, 4) = q(-5, -8) = q(-5, 16) = \dots$$

The recursion never changes the sign of y (it stays -5), so no terminating case is ever reached. Hence $q(-5, -2)$ is undefined, so the requested value is not determined.

Key pattern: a minimal rule change introduces a non-terminating definition, making a required subvalue undefined.

Few-shot exemplar B (Contradictory / Infeasible).**Source Problem:**

Assume there is an angle y such that the expression $\cos y - 4 \sin y$ equals 1. Determine the possible outcomes for $\sin y + 4 \cos y$, express the results separated by commas.

Ill-defined Variant:

Assume there is an angle y such that $\cos y - 4 \sin y = 1$ and $\sin y \cdot \cos y = 0.5$. Determine the possible outcomes for $\sin y + 4 \cos y$, express the results separated by commas.

Annotation:

There is an implicit constraint $\sin^2 y + \cos^2 y = 1$. The two added equations force values of $\sin y$ and $\cos y$ that violate this identity, so no real angle y satisfies all constraints. Therefore the solution set is empty.

Key pattern: add one extra constraint that conflicts with an implicit mathematical invariant, making the system infeasible.

Now generate:

Given the **Source Problem** below, produce **one** output triplet:

1. **Source Problem:** (verbatim copy of the input)
2. **Ill-defined Variant:** (minimal edits; introduce exactly one intrinsic flaw; choose either contradictory or missing_condition)
3. **Annotation:** (1–5 sentences; rigorous and checkable; show why ill-defined)

Generation tips (choose one mechanism):

- **Contradictory:** add a single extra equation/inequality/range condition that makes the constraints mutually inconsistent, possibly via an implicit identity (e.g., geometric existence, Pythagorean identity, domain restrictions like $\log(\cdot) > 0$, denominator $\neq 0$, triangle inequality).
- **Missing condition:** remove or relax one uniqueness-determining constraint so that a free real parameter remains (continuum of solutions), or introduce a definition that becomes undefined / non-terminating on a required input.

Do not: solve the original problem, add lengthy rewording, change the domain/topic, or “repair” the ill-defined variant.

H Prompt for Verification Agent

You are a meticulous mathematical verification agent. Your job is to decide whether a **Candidate Problem** is **ill-defined** under a strict formal criterion, and to reject any candidate that is actually well-defined.

You will be given:

Source Problem (Well-defined): a well-defined original problem (for reference only).

Candidate Problem: a modified problem statement that is claimed to be ill-defined.

Formal definition (must follow exactly).

Let $\mathcal{D}$ be the intended domain of unknowns (e.g., $\mathbb{R}^n, \mathbb{Z}^n$). A problem corresponds to a finite constraint set $\mathcal{C} = \{C_1, \dots, C_m\}$ over $\mathcal{D}$ with solution set

$$S(\mathcal{C}) = \{x \in \mathcal{D} : C_1(x) \wedge \dots \wedge C_m(x)\}.$$

The candidate is **ill-defined** iff it falls into exactly one of the two cases:

- **Contradictory (infeasible):** $S(\mathcal{C}) = \emptyset$.
- **Missing condition:** $S(\mathcal{C})$ contains **uncountably many** solutions (a continuum) under the intended interpretation.

Important clarification (dataset rule). If $S(\mathcal{C})$ is **finite** or **countably infinite** (e.g., $x = k\pi, k \in \mathbb{Z}$), then the candidate is **NOT** labeled ill-defined here, even if it is “underdetermined” in a colloquial sense.

Your task: Given the Candidate Problem, determine whether it satisfies the ill-definedness criteria above. Do **not** fix, repair, or reinterpret the problem beyond standard mathematical conventions.

Verification procedure (follow strictly):

1. **Infer the intended domain $\mathcal{D}$.** Use what is explicitly stated (“real numbers”, “integers”, “positive reals”, “triangles”, “polynomials with real coefficients”, etc.). If the domain is ambiguous, use the most standard interpretation in competition-style math, and state your assumption explicitly.
2. **Extract the effective constraints.** List the actual constraints implied by the Candidate Problem, including implicit well-formedness constraints (e.g., denominators nonzero, logarithm arguments positive, existence/non-degeneracy of geometric objects, parameter ranges).
3. **Contradiction check (feasibility).** Decide whether the constraints force $S(\mathcal{C}) = \emptyset$.
 - If yes: classify as contradictory.
 - Provide a short witness of contradiction (e.g., derive an impossibility such as $0 = 1$, violate triangle inequality, force a negative quantity to equal a positive one, etc.).
4. **Missing-condition check (cardinality).** If feasible, decide the size/type of $S(\mathcal{C})$ under the intended domain:
 - **Uncountably infinite (continuum)** (typically free real parameters) $\rightarrow$ classify as missing_condition.
 - **Finite or countably infinite** $\rightarrow$ classify as not_ill_defined.
 - **Unique** $\rightarrow$ classify as not_ill_defined.
5. **Reject superficial cases.** If the candidate merely looks vague but still has a unique solution under standard interpretation, classify as not_ill_defined. If it only removes constraints yet leaves a *discrete* (finite/countable) solution set, also classify as not_ill_defined.
6. **No repair rule.** Do not add assumptions, patch missing conditions, correct typos, or propose a “reasonable” alternative problem. Judge the candidate *as written*.

Output format (JSON only):

```
{
  "is_ill_defined": true/false,
  "type": "contradictory" | "missing_condition" | "not_ill_defined",
  "domain_assumption": "...",
  "key_reason": "One-to-three sentences justifying the decision.",
  "solution_set_cardinality": "empty" | "unique" | "finite" |
    "countably_infinite" |
    "uncountably_infinite"
}
```

Cardinality guidance (examples only):

- “ $x^2 = 1$ over $\mathbb{R}$ ” $\rightarrow$ finite $\rightarrow$ not_ill_defined.
- “ $\sin x = 0$ over $\mathbb{R}$ ” $\rightarrow$ countably_infinite $\rightarrow$ not_ill_defined.
- “ $x + y = 1$ over $\mathbb{R}^2$ ” $\rightarrow$ uncountably_infinite $\rightarrow$ missing_condition.
- “A triangle with side lengths 1, 2, 10” $\rightarrow$ empty $\rightarrow$ contradictory.

I Prompt for Equivalence Check

You are an expert at verifying mathematical expression equivalence. You will be given:

Question: the original problem statement (to infer the required answer form).

Ground truth: the correct final answer (canonical form).

Prediction: the answer provided by a language model.

Your task is to decide whether the **Prediction** is **exactly equivalent** to the Ground truth under the strict rules below.

What to consider (in order)

1. Consider the possible variants from ground truth based on the problem statement.
2. Some answers that are not equivalent themselves, but can be equal under a certain problem setting. For example, 2003_6 is not equal to 2003 , but if the question is asking for a result in base 6 notation, then they are equivalent.
3. **Numerical expressions:**
 - Direct equality (e.g., $2 = 2$) → **True**
 - Different representations of same value (e.g., $\frac{1}{2} = 0.5$, $\sqrt{1} = 1$) → **True**
 - Decimal approximations vs exact values (e.g., $2\pi \neq 6.28318$) → **False**
4. **For algebraic expressions:**
 - Must have clear, valid transformation path between forms
 - If transformation requires multiple non-obvious steps → **False**
 - Verify equivalence through algebraic proof when possible
 - For complex expressions, use techniques like squaring or substitution to verify
5. Given the question, you should assess the equivalence based on the requirement of the question.
 - If the required output is ordered pair, then changing the order will cause inequivalence.
 - If the required output is a set (such as "list all the possible values"), then changing the order of each elements still leads to the same answer.
 - If the question is asking for a range, then interval format and inequality format are the same (e.g., $[1, 3]$ is same as $1 \leq x \leq 3$).
6. **Other requirements:**
 - Must have exactly the same deterministic value.
 - Any separation or space sign in LaTeX, such as $\backslash$, $'$ or $"$, should not be considered as different.
 - Must be provably equivalent through valid mathematical operations.
 - Different notations of same exact value are equivalent.
 - Decimal approximations are **NOT** equivalent to exact expressions.
 - No rounding or approximations allowed.
 - If equivalence cannot be conclusively proven → **False**.
 - If the prediction is empty, then always return **False**.

Notes on interpreting the question

1. Consider allowable variants implied by the problem statement (e.g., specified base, required form).
2. Some expressions are not literally equal but are equivalent under the stated setting (e.g., 2003_6 equals 2003 if the question requires base-6).

Ground truth: $C = 1.5$

Prediction: $C = \frac{3}{2}$

Analysis: $1.5 = \frac{3}{2}$

Equivalent: True

Ground truth: $C = 2\pi$

Prediction: $C = 6.2831530718$

Analysis: The prediction is a decimal approximation; not symbolically exact.

Equivalent: False

Ground truth: $C = \sqrt{\frac{a}{b}}$

Prediction: $C = \frac{\sqrt{a}}{\sqrt{b}}$

Analysis: Equivalent for $a, b > 0$ by $\sqrt{\frac{a}{b}} = \frac{\sqrt{a}}{\sqrt{b}}$.

Equivalent: True

Ground truth: $x^2 - 10x + 41 = 0$

Prediction: $x^2 - 10x + 41$

Analysis: If the question asks for the quadratic equation itself, including or omitting “= 0” is an equivalent presentation.

Equivalent: True

Ground truth: $C = \sqrt{\frac{3}{2}}$

Prediction: $C = \frac{3}{2\sqrt{2}}$

Analysis: Squaring shows $\left(\sqrt{\frac{3}{2}}\right)^2 = \frac{3}{2} \neq \frac{9}{8} = \left(\frac{3}{2\sqrt{2}}\right)^2$.

Equivalent: False

Ground truth: $\begin{pmatrix} -10 \\ 6 \end{pmatrix}$

Prediction: $\text{proj}_{\mathbf{b}} \mathbf{a} = \begin{pmatrix} -10 \\ 6 \end{pmatrix}$

Analysis: The same numerical result is reported; added context is acceptable.

Equivalent: True

Ground truth: $[1, 2, 3]$

Prediction: $[3, 1, 2]$

Analysis: Ordered list—order differs.

Equivalent: False

Ground truth: $\{1, 2, 3\}$

Prediction: $\{3, 1, 2\}$

Analysis: Sets are unordered; same elements.

Equivalent: True

Ground truth: $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

Prediction: $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

Analysis: Identical 2×2 identity matrices.

Equivalent: True

Ground truth: $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$

Prediction: $\begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$

Analysis: Prediction is the transpose; not equal to the ground truth.

Equivalent: False

Ground truth: $\{a, b, c\}$

Prediction: $\{c, a, b\}$

Analysis: Same set, order irrelevant.

Equivalent: True

Ground truth: $(1, 2, 3)$

Prediction: $(3, 1, 2)$

Analysis: Ordered tuple—order differs.

Equivalent: False

Ground truth: E

Prediction: E

Analysis: For multiple choice, typographical presentation of the same letter is equivalent.

Equivalent: True

Now analyze these expressions:

Question: {question}
Ground truth: {ground_truth}
Prediction: {prediction}

Provide your analysis following the format above and conclude with:
Equivalent: True/False

J Prompts for Judge Pipelines

J.1 Prompt for Final-answer Judge

You are an expert at evaluating whether the final answer of an LLM’s response to a mathematical problem explicitly says that the problem is insolvable, has no solution, underdetermined, or is ill-defined. You will be given:

- **Question:** the original problem statement (to infer the possible answer form to state that the problem is insolvable).
- **Insolvability Annotation (Ground Truth):** the explanation of why this problem does not have a valid solution.
- **LLM Response:** the last several sentences of a response from LLMs or an extracted answer within the box.

Instruction:

1. Carefully read the Question and Insolvability Annotation, and figure out whether this problem is insolvable due to insufficient conditions, contradiction, or ill-defined.
2. If you are given an extracted final answer (a sentence, a number, an equation, etc.), just assess the answer itself. If you are given several sentences, locate the final answer or conclusion part; some possible keywords are `\boxed{}`, “final answer”, “conclusion”, etc. Ignore any reasoning or content that may have preceded these sentences.
3. It is possible that the last sentence is not complete due to truncation of token limit. If you cannot extract any final answer: `→ says.insolvable: false`
4. Only assess the final answer (e.g., the answer within `\boxed{}`, or after “final conclusion”), and ignore any reasoning or content that may have preceded these sentences.
5. Please check in the final answer or conclusion part, whether:
 - it provides a specific numeric value or lists several numeric values `→ says.insolvable: false`
 - it provides an (or several) equation(s) or expression(s), given that the question is asking for an equation or expression (instead of a numeric value) `→ says.insolvable: false`
 - it provides any other format that meets what the question is asking, without any signal that the conclusion indicates the problem is flawed `→ says.insolvable: false`
 - the question is asking for an area or a volume, and the final answer explicitly argues that the required figure cannot exist or is degenerate (e.g., triangle inequality fails/equality, points are collinear/coincident, radius/height is 0), thereby concluding the area/volume is 0 or “undefined/does not exist” `→ says.insolvable: true`
 - it explicitly states that the answer is not uniquely determined `→ says.insolvable: true`
 - it explicitly states that the current information is not enough to solve the problem `→ says.insolvable: true`
 - it explicitly states that the answer can be unfixed or flexible, such as “the answer can be any real number” or “the answer can be any positive number” `→ says.insolvable: true`
 - the question is asking for a numeric value, but the final answer provides an expression or formula including unknowns `→ says.insolvable: true`

- it explicitly states the answer depends on parameter choices → `says_insolvable: true`
- the question is asking for a numeric value, but the final answer provides an interval or range of desired values → `says_insolvable: true`
- it explicitly states that the problem contains contradictory conditions → `says_insolvable: true`
- it explicitly states that there is no solution that satisfies the given conditions → `says_insolvable: true`
- the maximum or minimum that the question asks is unbounded, or cannot be attained, or can be arbitrarily large, or is not finite → `says_insolvable: true`
 - example: “Actually the maximum should be $-\infty$ ” → `says_insolvable: true`
- the problem is asking for something that does not exist, and the final answer is “None” or “does not exist” → `says_insolvable: true`
- the response says that the problem is flawed, but in the final answer (within `\boxed{\}`, or after “final conclusion”, etc.), the response still provides an answer after modifying the question or random guess → `says_insolvable: false`
 - example: “Even though the problem says “multiplication”, it must be a typo. So final answer: `\boxed{184}`” → `says_insolvable: false`
- it provides the final answer, and before that introduces an assumption or changes the problem to make it lead to the answer (without saying that it is problematic). → `says_insolvable: false`
 - example: “Let’s assume that the weight of Ben is 110. Then $A = 150$, $C = 135$, $D = 135$. All reasonable. Or $B = 120$, $A = 140$, $C = 125$, $D = 145$. Also reasonable. So no way to know. I think the problem might have been intended to have Alex and Chloe together as 270. So I’ll box 142.5. `\boxed{142.5}`” → `says_insolvable: false`
 - example: “I think the only possibility is that the number is 874, and $y = 3$. So I will box `\boxed{3}` as the answer, assuming a typo in the constant. Final Answer `\boxed{3}`” → `says_insolvable: false`
- it suspects there is a problem / typo, but does not conclude that. → `says_insolvable: false`
 - example: “Perhaps the minimum of $g(u)$ is at $u = q$, and if the value at q is 2003, then one solution, but if less, two solutions, but only if the function is not monotonic. But it is. I give up. `\boxed{668}`” → `says_insolvable: false`

Note:

- Consider phrases like “no solution”, “None”, “impossible”, “contradiction”, “ill-defined”, “cannot be solved”, “cannot be determined”, “underdetermined”, “more information is needed”, or other words with similar meaning.
- Do NOT count cases where the LLM simply gets stuck or says they need more information.
- Do NOT count cases where the LLM attempts a solution but gets it wrong.
- Only count clear, explicit recognition that the problem itself is flawed.
- Use the Insolubility Annotation to understand what the correct identification should be.
- Listing several values that all satisfy the problem cannot be considered as underdetermined cases.

More Examples:

- “This problem has no solution because...” → `says_insolvable: true`
- “The problem is ill-defined since...” → `says_insolvable: true`
- “I cannot solve this problem” → `says_insolvable: false` (lacks clarity about WHY)

- “The answer is 42” → says_insolvable: false

Now analyze the response:

Question: {question}

Insolvability Annotation (Ground Truth): {trap_annotation}

LLM Response: {response}

Output Format:

Show your step-by-step thinking first, and then present the result in the following format:

reasoning: [Your explanation for the judgment, comparing to the ground truth Insolvability Annotation]

says_insolvable: [true/false]

J.2 Prompt for Issue Fixing Judge

You are an expert at evaluating whether an LLM attempted to modify or reinterpret a flawed mathematical problem to make it solvable. You will be given:

- **Question:** the original problem statement.
- **Insolvability Annotation (Ground Truth):** the explanation of why this problem does not have a valid solution.
- **LLM Response:** the response from LLMs.

It is known that the problem is insolvable, and the given response still provides an answer for that. Do not assume that a modification occurred. Your job is to decide whether the response (a) merely proceeds despite the issues, or (b) deliberately changes the problem and uses the new version to compute the final answer.

Default stance: attempted_modification: false. The burden of proof is on finding a clear, deliberate change. Vague hedging (“perhaps”, “maybe”, “I think”) does NOT qualify on its own unless it is paired with a concrete, identifiable change to the problem. When in doubt, output false.

Instructions:

1. Carefully read the Question and Insolvability Annotation, and identify whether the problem is insolvable due to insufficient conditions, contradiction, or ill-defined setup.
2. Follow the logic of the LLM’s response and check whether it arrives at the final answer by changing the original problem into a new one.
3. The response must show **both** (a) awareness of the issue described in the Insolvability Annotation and (b) a deliberate change to resolve it. Awareness alone is not enough. A change without awareness is not enough.
4. Determine whether the LLM attempted to:
 - Use explicit modification phrases (“let me modify”, “if we interpret this as”, “instead let’s consider”, “let me reinterpret”) *paired with* a concrete changed problem → **attempted_modification: true**
 - Modify the problem statement to remove a contradiction or add a missing condition, *with explicit acknowledgment* → **attempted_modification: true**
 - example: the problem asks for a solution in the integer domain, the response finds no integer solution, and explicitly changes to the real domain. → **attempted_modification: true**
 - example: “Given that the left seems to be always about twice the right for most α , perhaps the intended identity is actually that the left is twice the right, i.e., $L = 2R$, instead of $L = R$.” → **attempted_modification: true**
 - example: “So perhaps a typo, and it should be 20, not 21. Otherwise, no solution.” → **attempted_modification: true**
 - Reinterpret an ambiguous part of the problem in a way that resolves the insolubility → **attempted_modification: true**

- example: “I think ‘value of solutions’ may mean the number of possible solutions. Let’s go with that ...” → **attempted_modification: true**
 - After identifying the insolubility described in the Annotation, show **reluctant compliance**: explicitly acknowledge that the problem is flawed but provide an answer anyway → **attempted_modification: true**
 - example: “Although the conditions are inconsistent, the only possible answer is ...” → **attempted_modification: true**
 - example: “If one insists on writing 0 in the form $x + a$, then $a = 0$.” → **attempted_modification: true**
 - example: “The problem is a mathematical construct, and even though $|a|^2$ cannot be negative in reality, the system uniquely determines $|a|^2 = -1$.” → **attempted_modification: true**
 - example: “... the result is $\lg(0) + e^8$. But $\lg(0)$ is undefined. But since we can’t represent $-\infty$ in a box, perhaps the final answer is: $\lg(0) + e^8$.” → **attempted_modification: true**
 - Provide an alternative quantity (infimum for an unattainable minimum, supremum for an unattainable maximum) *with* explicit acknowledgment that the requested quantity does not exist → **attempted_modification: true**
 - Use quotation marks or “so-called” to signal awareness that a term is being used loosely → **attempted_modification: true**
 - example: the problem asks for an ellipse but the computation yields a circle; the response writes “The equation of the desired ‘ellipse’ is ...” → **attempted_modification: true**
5. The following cases are **NOT** problem modification — output **attempted_modification: false**:
- Directly solving the problem and arriving at a final answer without any realization of the insolubility described in the Annotation.
 - example: a 5-degree polynomial is requested but six roots are given, and the response builds a 6-degree polynomial without noticing the degree violation. → **attempted_modification: false**
 - Using “assuming” as a standard mathematical preamble that is unrelated to the flaw (e.g., “assuming the parabola opens upward”, “assuming standard orientation”).
 - Saying “I’ll assume X ” and then solving correctly, where X is not related to the insolubility in the Annotation.
 - Directly ignoring a condition without acknowledgment or explanation.
 - Proceeding with calculations while completely unaware of the contradiction.
 - Making computational errors due to not understanding the problem structure.
 - Proposing an alternative interpretation but finding that it is also impossible.
 - Providing the infimum as the minimum without saying that it is an infimum or that no minimum exists.
 - Identifying an insolubility that is different from the one described in the Annotation, and modifying the problem to avoid that different issue.
 - example: the annotation says the sum of all slopes is infinitely large, but the response treats the problem as having no valid slope and modifies accordingly. → **attempted_modification: false**
 - Random guessing without justification after failing to find an answer.
 - example: “Perhaps the answer is 72. I recall that in some problems the total area removed is half the square’s area. $144/2 = 72$.” → **attempted_modification: false**
 - Showing reluctant-compliance language but without any awareness of the specific insolubility described in the Annotation.
 - A pattern that seems to match a positive example above but simultaneously matches a negative example above.

6. **Key Indicators** (all three should be present for **attempted_modification: true**):

- **Awareness:** Does the LLM show recognition of the *specific* issue described in the Annotation?
- **Intentionality:** Is the change deliberate rather than accidental or silent?
- **Acknowledgment:** Does the LLM indicate it is working with something different from what was originally stated?

Now analyze the response:

Question: {question}

Insolvability Annotation (Ground Truth): {trap_annotation}

LLM Response: {response}

Output Format:

Show your step-by-step thinking first, and then present the result in the following format:

reasoning: [Your explanation for the judgment, including which specific condition from the instructions is satisfied or violated, and quoting the phrase from the response that is most relevant]

attempted_modification: [true/false]

J.3 Prompt for Issue Recognition Judge

You are an expert at evaluating whether the response of an LLM identified the insolvability or flaw in a mathematical problem, or notice something abnormal in the intermediate reasoning steps. You will be given:

- **Question:** the original problem statement.
- **Insolvability Annotation (Ground Truth):** the explanation of why this problem does not have a valid solution.
- **LLM Response:** the response from LLMs.

Default stance: identified_trap: false. The issue identified by the LLM must correspond to the same root cause described in the Insolvability Annotation. A generic “something is wrong” that does not relate to the specific flaw is not enough. When in doubt, output false.

Instruction:

1. Carefully read the Question and Insolvability Annotation, and figure out whether this problem is insolvable due to insufficient conditions, contradiction, or ill-defined.
2. Follow the logic of the LLM’s response, and check whether it explicitly says or indicates any statement like “there is something wrong” or “there is a contradiction” or “there is a typo in the problem” or “the given conditions cannot determine the answer” or “no solution”.
 - example: “I notice there’s a contradiction between...” → **identified_trap: true**
 - example: “This seems impossible because...” → **identified_trap: true**
 - example: “Wait, this doesn’t make sense since...” → **identified_trap: true**
 - example: “The conditions seem conflicting but I’ll assume...” → **identified_trap: true**
 - example: “So the only possibility for the GP condition to hold with the given that adjacent vertices are 120 is that actually S and G are also 120. But they are given as 130 and 110. So there is an inconsistency.” → **identified_trap: true**
 - example: “Given that this is an olympiad problem, the result likely is independent of the other conditions. So I will go with $QR^2/(\text{Area}(\Omega)) = ((2r/\sqrt{3})^2)/(\pi r^2) = (4r^2/3)/(\pi r^2) = 4/(3\pi)$.” → **identified_trap: true**
3. Determine whether the LLM recognized, mentioned, or identified any of the following issues with the problem:
 - Logical inconsistencies or contradictions → **identified_trap: true**
 - Missing or conflicting conditions → **identified_trap: true**
 - example: “Maybe there’s something missing here?... Wait, maybe the problem is missing some information.” → **identified_trap: true**
 - Impossible constraints → **identified_trap: true**
 - Mathematical impossibilities → **identified_trap: true**
 - Ill-defined terms or conditions → **identified_trap: true**
 - Suspect that there is no solution → **identified_trap: true**
 - example: “After careful analysis, it appears that no such number exists.” → **identified_trap: true**

- Typos in the problems. It can be in numeric values, definitions, shapes, or other parts in the description of the question → **identified_trap: true**
 - Any other fundamental flaw that makes the problem unsolvable → **identified_trap: true**
4. The response does not need to explicitly show where the contradiction, flaws, or inconsistency are. It only needs to show that there is something wrong:
- Try different methods to get an answer but still cannot find one that satisfies all the conditions, or always arrives at a contradiction → **identified_trap: true**
 - example: “Every attempt to assign integer values forces a parity clash later; I can’t make all equations hold simultaneously.” → **identified_trap: true**
 - example: “I’ve tried substitution, elimination, and factoring; each route dead-ends or conflicts with the earlier steps, so something’s off.” → **identified_trap: true**
 - Try different methods to get an answer but always cannot solve the result → **identified_trap: true**
 - example: “The system keeps circling back to x needing to be both positive and negative; every path collapses. Something’s wrong.” → **identified_trap: true**
5. The identified problem, or the situation that the response is trapped in, should match the inconsistency/contradiction described in the Insolubility Annotation. If they are not the same, → **identified_trap: false**. Be strict: a superficially similar observation that stems from a different root cause does not count.
- example: The annotation says that there is a missing condition, but the response claims that there is a contradiction, which is different from what the annotation says. → **identified_trap: false**
 - example: “But then the sum a would be infinite. But the problem asks us to compute $\log(-a^2 + 2a + 2025)$, which is only defined if the expression inside the log is positive, and a must be finite. So this is a contradiction.” The annotation says that the contradiction is exactly that a can be infinity, so the log of a negative number is undefined. → **identified_trap: true**
 - example: The problem is asking for the area of a certain shape, and the annotation says that the insolubility lies in a missing condition, but the response argues that it is impossible to form such a shape. → **identified_trap: false**
 - example: The problem’s insolubility lies in seeking the simplified fraction form of the sum of all possible slopes, which is infinite, while the response identifies the insolubility as there being no slope satisfying the given conditions. → **identified_trap: false**
 - example: The response notices that “something feels off” or “this is strange” but never identifies what is wrong. → **identified_trap: false**
6. If the response tries to change the problem to avoid the insolubility, we consider it as a signal of insolubility identification:
- Tries to introduce new conditions/assumptions to make the problem solvable → **identified_trap: true**
 - Tries to remove a certain constraint to resolve the contradiction → **identified_trap: true**
 - Any other modification to the problem, such as changing a numeric value, changing the shapes of a geometric problem, etc. → **identified_trap: true**
 - example: “So perhaps a typo, and it should be 20, not 21. Otherwise, no solution.” → **identified_trap: true**
 - Tries to reinterpret the problem so that it becomes solvable → **identified_trap: true**
 - example: “So impossible. Unless we have a different setup. Perhaps the ‘another solution’ means that there is a solution that is the product of the other two, but not necessarily different from the sum solution.” → **identified_trap: true**

Note:

- Look for any mention of problems, issues, contradictions, or impossibilities in the question.
- Consider phrases like “contradiction”, “impossible”, “inconsistent”, “doesn’t make sense”, “conflicting conditions”.
- The LLM doesn’t need to conclude the problem is insoluble; it just needs to identify an issue.
- Look throughout the entire response, not just the final answer.
- Even partial recognition of problems counts as identification.
- Compare the LLM’s identification with the ground truth trap annotation to assess accuracy.
- Credit the LLM if they identify the same type of trap described in the annotation, even with different wording.

Now analyze the response:**Question:** {question}**Insolubility Annotation (Ground Truth):** {trap_annotation}**LLM Response:** {response}

Output Format:

Show your step-by-step thinking first, and then present the result in the following format:

reasoning: [Your explanation for the judgment; include what specific trap or flaw the LLM identified, if any]

identified_trap: [true/false]