VCSearch: Bridging the Gap Between Well-Defined and Ill-Defined Problems in Mathematical Reasoning

Anonymous ACL submission

Abstract

Large language models (LLMs) have demonstrated impressive performance on reasoning tasks, including mathematical reasoning. However, the current evaluation mostly focuses on carefully constructed benchmarks and neglects the consideration of real-world reasoning problems that present missing or contradictory conditions, known as ill-defined problems. To further study this problem, we develop a large-scale benchmark called ***P**roblems with **M**issing and **C**ontradictory conditions* (PMC) containing over 5,000 validated ill-defined mathematical problems. Our preliminary experiments through PMC reveal two challenges about existing methods: (1) traditional methods exhibit a trade-off between solving accuracy and rejection capabilities, and (2) formal methods struggle with modeling complex problems. To address these challenges, We develop ***V**ariable-**C**onstraint **Search*** (VCSEARCH), a training-free framework that leverages formal language to detect ill-defined problems, where a variable-constraint pair search strategy is incorporated to improve the modeling capability of formal language. Extensive experiments demonstrate that VCSEARCH improves the accuracy of identifying unsolvable problems by at least 12% across different LLMs, thus achieving stronger robust mathematical reasoning ability.

1 Introduction

Large language models (LLMs) have demonstrated strong performance on various reasoning tasks, including commonsense (Zhao et al., 2023), quantitative (Lewkowycz et al., 2022), and visual reasoning (Gupta and Kembhavi, 2023). Mathematical problem solving (Cobbe et al., 2021) serves as a fundamental benchmark for evaluating LLMs' reasoning capabilities (Ahn et al., 2024). Recent advances in prompt-based methods (Wei et al., 2022; Ye et al., 2024) and fine-tuning approaches (Yu et al., 2023; Li et al., 2024b) have significantly improved their mathematical reasoning capabilities.

Although existing studies have improved the performance of LLMs on well-defined mathematical benchmarks (Cobbe et al., 2021; Patel et al., 2021), they often overlook a critical challenge in real-world applications: the ability to reject ill-defined problems (Zhao et al., 2024). These problems, which contain missing or contradictory conditions (Puchalska and Semadeni, 1987), are particularly common in educational scenarios. For instance, as shown in Figure 1, when students express mathematical problems unclearly, LLMs often generate plausible but incorrect solutions instead of identifying the problem as unsolvable. Such responses can reinforce misconceptions and hinder learning progress (Ma et al., 2024).

However, most existing benchmark about math reasoning robustness (Shi et al., 2023; Zhou et al., 2024) focus on whether the model can still answer the question in the presence of interference, lacking a systematic evaluation of the model's ability to recognize and reject ill-defined problems. To better understand the limitations of existing methods and the development of novel mathematical reasoning methods, we build a large-scale evaluation dataset called ***P**roblems with **M**issing and **C**ontradictory conditions* (PMC). This dataset contains over 5,000 validated ill-defined mathematical problems for comprehensive evaluation.

Our preliminary experiments reveal two major challenges when handling ill-defined problems. First, traditional methods, e.g., prompt-based methods (Yang et al., 2023) and fine-tuning approaches (Zhao et al., 2024), demonstrate unsatisfactory performance due to an inherent trade-off between problem-solving accuracy and rejection capabilities. Second, although formal methods (Ye et al., 2024; Pan et al., 2023) offer unified problem-solving and rejection capabilities, they struggle to accurately model complex problems in formal language.

To address these challenges, we propose VC-

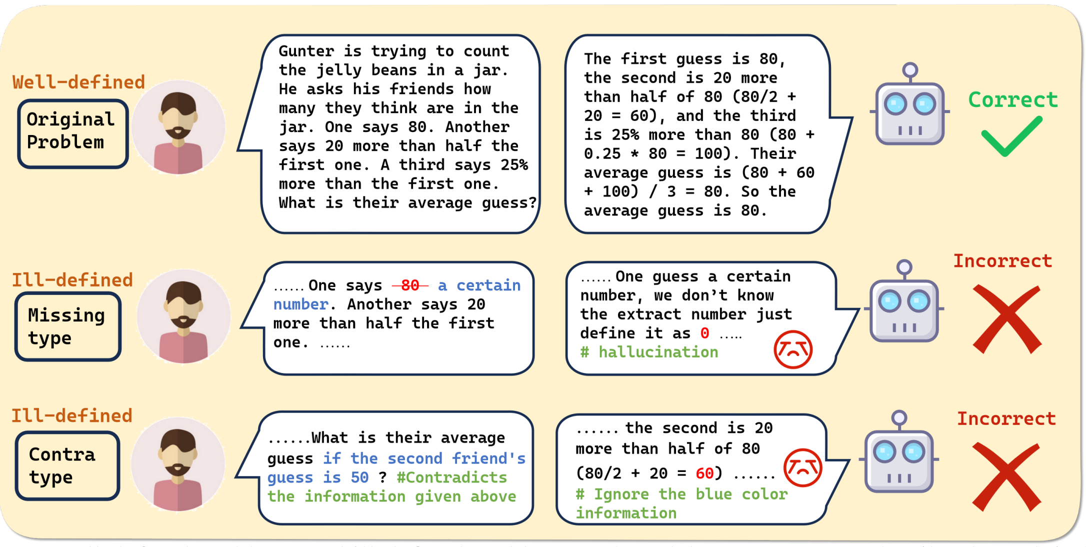


Figure 1: Well-defined problems and ill-defined problems and model's response. (Red strike-through indicates deleted sentences, blue indicates added sentences and green indicates explanation)

SEARCH (*Variable-Constraint* ***Search***), a training-free framework that systematically detects ill-defined problems through formal language to address the challenge of trade-offs. The key innovation of VCSEARCH lies in its variable-constraint dynamic search mechanism, which decomposes complex problems that are hard to model into dynamically extensible variable-constraint pairs, implementing an iterative optimization strategy where discovered variables guide constraint generation and existing constraints inform variable identification. Experimental results demonstrate that VCSEARCH achieves an at least 12% improvement in rejection accuracy for unsolvable problems compared to state-of-the-art methods, thus achieving stronger robust mathematical reasoning ability in realistic scenarios. Our main contributions can be summarized as follows:

1) We introduce a practical problem of evaluating mathematical reasoning robustness and present PMC, a large-scale benchmark dataset containing over 5,000 validated ill-posed mathematical problems.

2) We develop VCSEARCH, a training-free framework that leverages formal language to detect ill-defined problems, where a variable-constraint pair search strategy is incorporated to improve the modeling capability of formal language.

3) Extensive experiments demonstrate that VCSEARCH improves the accuracy of identifying unsolvable problems by at least 12% across different LLMs, thus achieving stronger robust mathematical reasoning ability in realistic scenarios.

2 PMC Benchmark and Analysis

In this section, we first introduce our PMC benchmark, which consists of two types, i.e., Contra-type and Missing-type, by mutating problems from four common math datasets. Then, our analysis presents the challenges of rejecting ill-defined problems and the limitations of existing methods.

2.1 Benchmark Construction

We choose four common mathematical reasoning datasets, that is, GSM8k (Cobbe et al., 2021), SVAMP (Patel et al., 2021), AddSub (Hosseini et al., 2014), and MultiArith (Koncel-Kedziorski et al., 2016), as seed datasets to construct PMC. We define the problems in the seed dataset as **well-defined** problems, meaning that the given conditions in the problem statement are sufficient to derive a unique solution. In contrast, the problems we aim to construct are **ill-defined** problems, where the given conditions are insufficient—either due to missing necessary constraints or internal contradictions—making the problem unsolvable.

Our construction methodology employs a prompting-based strategy with Large Language Models (LLMs). Initially, the LLM is prompted to decompose a seed problem and ascertain all pertinent variables. Subsequently, the model is instructed to implement targeted modifications to the original problem conditions. To generate "missing-type" problems, a numerical value within a specific constraint is substituted with an indeterminate term, thereby rendering the problem definition incomplete. For "contra-type" problems, contradictory constraints pertaining to the variables are introduced, yielding problems that are inherently self-contradictory and thus pathological. To verify the unsolvable (ill-defined) nature of the constructed problems, we utilize a panel of heterogeneous LLMs (e.g., Deepseek-V3 (Liu et al., 2024), Doubao, and GLM (Zeng et al., 2024)) to assess whether the modified problem possesses a unique solution. A problem is classified as unsolvable if a consensus is reached among all participating LLMs that no solution exists. In instances where any model deems the problem solvable, human annotators are engaged to meticulously review the problem and confirm its unsolvable status.

Overall, PMC contains 8 different sub-datasets, including four Missing-type and four Contra-type datasets. An illustration of mutated problems of PMC is presented in Fig 1, and more detailed information about PMC (construction prompt, examples, etc.) can be found in the appendix.

2.2 Evaluation Protocol

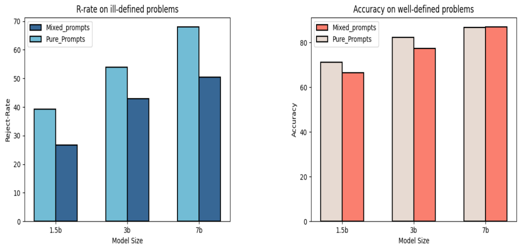
To evaluate the robustness of methods in mathematical reasoning problems with missing and contradictory conditions, we introduce two evaluation metrics: the Rejection Rate (R-Rate) and the Reaction Score (R-Score). R-Rate quantifies a method's ability to identify ill-defined problems. R-Score evaluates a method's overall performance in both handling ill-defined problems and solving well-defined problems.

For a well-defined dataset $\mathcal{D}_w$, let $\mathcal{D}_i$ be its ill-defined counterpart. For any problem p, let $g(p)$ denote its ground truth solution, where $g(p) = \text{Reject}$ for ill-defined problems. Let $f(p)$ denote the solution generated by a method, where $f(p) = \text{Reject}$ indicates the method rejects to solve p. We define the R-Rate and R-Score as follows:

Rejection Rate. R-Rate is the percentage of ill-defined problems correctly rejected by method $f(\cdot)$:

$$\frac{\sum_{p \in \mathcal{D}_i} \mathbb{I}\left[f(p) = \text{Reject}\right]}{|\mathcal{D}_i|} \tag{1}$$

Reaction Score. R-Score measures a method's overall performance by considering three scenarios: (a) correctly rejecting ill-defined problems, (b) correctly solving well-defined problems, and (c) rejecting well-defined problems. A method receives one point for scenarios (a) and (b), and 0.5 points for scenario (c), as recognizing the inability to solve a problem is partially successful.



(a) ill-defined problems (b) well-defined problems

Figure 2: Trade-off of traditional methods

$$\Big(\sum_{p \in \mathcal{D}_i} \mathbb{I}[f(p) = \text{Reject}] + \sum_{p \in \mathcal{D}_w} \mathbb{I}[f(p) = g(p)] + 0.5 \sum_{p \in \mathcal{D}_w} \mathbb{I}[f(p) = \text{Reject}] \Big) / (|\mathcal{D}_i| + |\mathcal{D}_w|) \tag{2}$$

2.3 Problem Analysis

We conduct a series of preliminary experiments on the PMC benchmark testing platform (with more detailed experimental modules to be elaborated in subsequent sections). The results are shown in Figure 2. We use "pure prompt" to refer to directly prompting the model to solve well-defined or ill-defined problems (focusing on one type), and "mixed prompt" to denote prompting the model to solve mathematical problems, where the model is instructed to reject if it deems the problem unsolvable. We observe that the base model exhibited certain problem-solving and rejection capabilities. However, there is a significant conflict between these two abilities: when the model is required to solve a problem while simultaneously employing a rejection mechanism, both its rejection and problem-solving capabilities are notably limited. This suggests a trade-off between the two and this trade-off becomes more pronounced as the model size decreases.

3 Methodology

To address the trade-off between solving accuracy and rejection capabilities, we propose a novel framework called VCSEARCH. This training-free framework leverages formal language modeling capabilities to detect ill-defined problems and enhances existing mathematical reasoning methods with the ability to identify unsolvable problems. However, modeling mathematical problems with

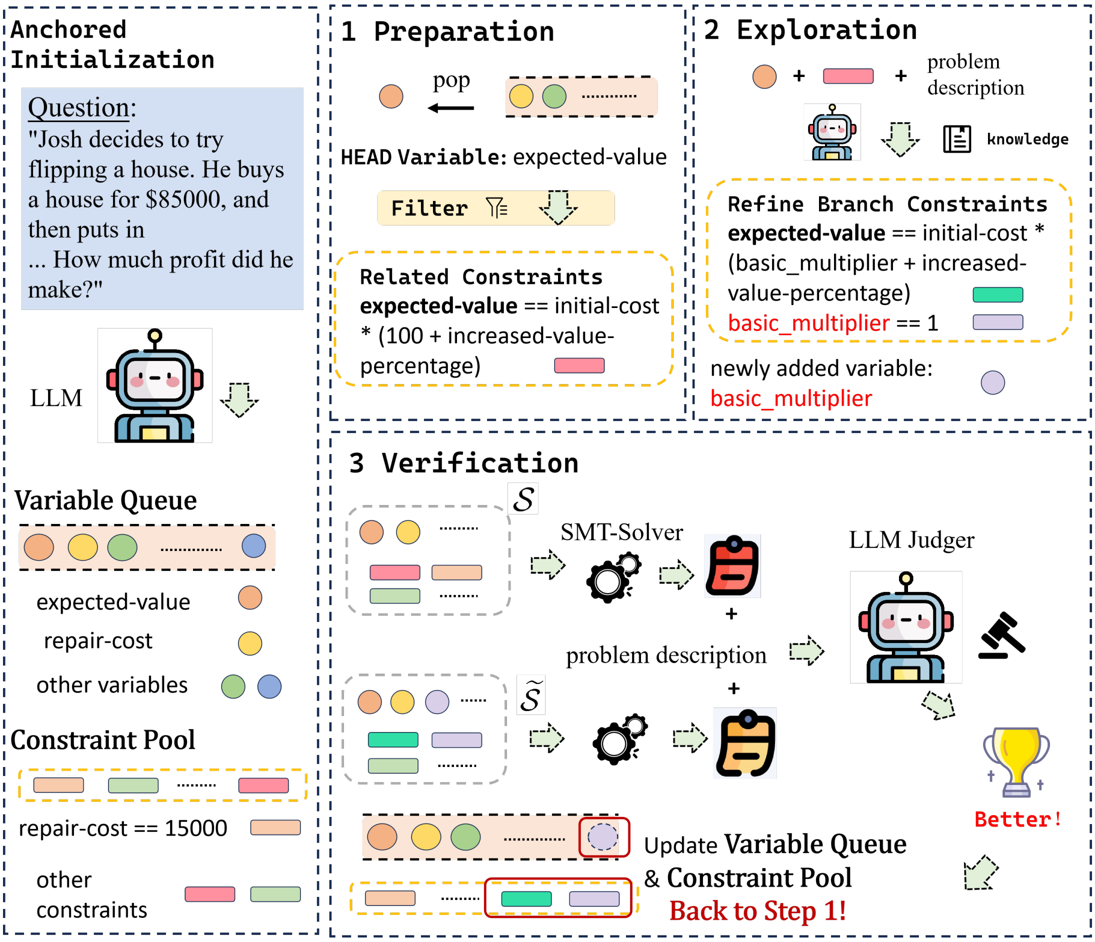


Figure 3: An overview of VCSEARCH. The left panel illustrates the outcome of the successful initialization phase, culminating in an initialized draft formal modeling state, denoted as S. Within this state representation, individual dots correspond to variables v, while elongated rectangles signify constraints c. Conversely, the right panel depicts the iterative process of \algo. Each iteration commences with the extraction of a head variable, followed by the sequential execution of three distinct steps: (1) Preparation, (2) Exploration, and (3) Verification.

formal language accurately is not trivial, directly using formalized examples as context prompts did not yield optimal results(in Table 1), raising the following challenge: LLMs fail to model problems with formal language accurately in one pass. How can we improve the problem modeling ability?

To tackle this challenge, we first propose a *Variable-Constraint Dynamic Search* that systematically discovers new variables and constraints through an iterative searching process consisting of three steps: Preparation, Exploration, and Verification. Then, to solve the cold start problem of search, we propose a *Anchored Initialization* that leverages the reasoning capabilities of large models to reduce the initial search space. We use SMT-Lib (Barrett et al., 2010) as the formal modeling language and Z3 (de Moura and Bjørner, 2008) as the formal solver in our approach and the overall framework is shown in Figure 3.

3.1 Variable-Constraint Dynamic Search

LLMs have limitations in precisely modeling complex problems with formal language in a single pass due to the multiple variables and constraints involved which increase the modeling difficulty. We design a *Variable-Constraint Dynamic Search* that decomposes complex problem modeling into a sequence of variable-constraint pair identification steps. This approach enables an iterative search that progressively improves the formal modeling.

To achieve this, we implement the *Variable-Constraint Dynamic Search* containing three systematic steps, i.e., Preparation, Exploration, and Verification. In each iteration, we perform the above four processes on the extracted variable. For problem p, we denote the modeling state as $\mathcal{S} = (\mathcal{V}, \mathcal{C})$ where $\mathcal{V}$ is the set of variables and $\mathcal{C}$ is the set of constraints corresponding to $\mathcal{V}$.

Preparation Step. This step selects a single variable and its associated constraints from $\mathcal{S}$ to reduce the complexity of the constraint analysis process, rather than considering all variables and constraints at once. Given the variable set $\mathcal{V}$ and constraint set $\mathcal{C}$, we select one unexplored variable from the set $\mathcal{V}$ as the head variable v_h and extract its related constraints $\mathcal{C}_h$ from $\mathcal{C}$:

$$\mathcal{C}_h = \{c \mid v_h \in \text{vars}(c) \text{ and } c \in \mathcal{C}\} \tag{3}$$

where $\text{vars}(\cdot)$ returns the set of variables in a given constraint, and c represents a constraint from $\mathcal{C}$.

Exploration Step. This step explores new constraints and variables with the help of implicit knowledge from the LLM to improve the problem modeling. Specifically, we prompt the LLM to generate the polished constraints $\widetilde{\mathcal{C}}_h$, relating to variable v_h for current problem p:

$$\widetilde{\mathcal{C}}_h = \text{LLM}_E(p,\, v_h,\, \mathcal{C}_h) \tag{4}$$

where LLM_E is denoted as the LLM prompted for exploration. The newly identified variables $\widetilde{\mathcal{V}}_h$ are

$$\widetilde{\mathcal{V}}_h = \{v \mid v \in \text{vars}(\widetilde{\mathcal{C}}_h) \text{ and } v \notin \mathcal{V}\}. \tag{5}$$

Verification Step. After exploring new constraints and variables, we can build a new problem modeling $\widetilde{\mathcal{S}}$ as follows.

$$\widetilde{\mathcal{S}} = \left(\mathcal{V} \cup \widetilde{\mathcal{V}}_h,\, (\mathcal{C} \setminus \mathcal{C}_h) \cup \widetilde{\mathcal{C}}_h\right) \tag{6}$$

where the new variables are added at the tail of original variable set $\mathcal{V}$ and the polished constraints replaced the original related constraints in the constraint set $\mathcal{C}$. Then, a SMT solver Φ is adopted to solve the problem modeling state $\widetilde{\mathcal{S}}$ and yield a solution $\widetilde{\mathcal{R}} = \Phi(\widetilde{\mathcal{S}})$. Inspired by LLMs as a judge (Zheng et al., 2023; Huang et al., 2024), we compare the original problem modeling $\mathcal{S}$ with its solution $\mathcal{R} = \Phi(\mathcal{S})$ and the new problem modeling state $\widetilde{\mathcal{S}}$ with the solution $\widetilde{\mathcal{R}}$ as follows:

$$\widetilde{\mathcal{S}}^* = \text{LLM}_J\left(p, (\mathcal{S}, \mathcal{R}),\, (\widetilde{\mathcal{S}}, \widetilde{\mathcal{R}})\right) \tag{7}$$

where LLM_J is denoted as the LLM prompted for verification and $\widetilde{\mathcal{S}}^*$ is the selected state from new state $\widetilde{\mathcal{S}}$ and original state $\mathcal{S}$. Finally, we replace current state $\mathcal{S}$ with selected state $\widetilde{\mathcal{S}}^*$ for the subsequent process and add newly detected variable to the variable queue $\mathcal{V}$. This repeated searching process is terminated until all variables in $\mathcal{V}$ are explored.

This step not only ensures the adaptive nature of the search process but also effectively leverages the reasoning capabilities of LLMs to gradually improve problem modeling $\mathcal{S}$.

3.2 Anchored Initialization

However, the search process is particularly challenging at the outset due to the difficulty in initializing the search state, as the initial state contains limited information. The search space is vast, and without a reliable initialization, it is challenging to converge to a valid state. This can result in the model being overly conservative, leading to the rejection of many well-defined problems(Table 4).

To address this challenge, we propose a *Anchored Initialization* that leverages the reasoning capabilities of the LLM to generate a preliminary anchor state $\widehat{\mathcal{S}}$ as an anchored initialization state for *Variable-Constraint Dynamic Search*.

Specifically, we first prompt the LLM to generate a draft modeling state $\hat{\mathcal{S}} = (\widehat{\mathcal{V}}, \widehat{\mathcal{C}})$ for problem p:

$$(\widehat{\mathcal{V}}, \widehat{\mathcal{C}}) = \text{LLM}_I(p) \tag{8}$$

where LLM_I is denoted as the LLM prompted for initialization with four examples in the context. Then, we adopt a SMT solver Φ compute the solution $\widehat{\mathcal{R}} = \Phi(\widehat{\mathcal{S}})$ of the draft modeling state $\widehat{\mathcal{S}}$ for validation. If the solution $\widehat{\mathcal{R}}$ is valid, we regard the draft modeling state $\widehat{\mathcal{S}}$ as the initialization state $\mathcal{S}$ for *Variable-Constraint Dynamic Search*. Otherwise, we only adopt the variable set $\widehat{\mathcal{V}}$ and empty constraint set as the initialization state $\mathcal{S}$ for subsequent searching.

$$\mathcal{S} = \begin{cases} (\widehat{\mathcal{V}}, \widehat{\mathcal{C}}) & \text{if } \Phi(\widehat{\mathcal{S}}) \neq \varnothing, \\ (\widehat{\mathcal{V}}, \varnothing) & \text{if } \Phi(\widehat{\mathcal{S}}) = \varnothing. \end{cases} \tag{9}$$

This module effectively incorporates the reasoning capabilities of the LLM to reduce the complexity of the search space at the beginning of the searching by providing a reliable initial anchor.

3.3 Integration with Existing Methods

The VCSEARCH framework finally returns a problem modeling state $\mathcal{S}^* = (\mathcal{V}^*, \mathcal{C}^*)$, and its solution can be computed by a SMT solver Φ, i.e, $\mathcal{R}^* = \Phi(\mathcal{S}^*)$. Therefore, we can integrate the VCSEARCH with any existing methods to enhance their ability to reject ill-defined problems. Specifically, we first verify the $\mathcal{R}^*$ set is valid by the VCSEARCH and the SMT solver. If $\mathcal{R}^*$ is valid, we regard the problem is well-defined and call existing methods to solve it. Otherwise, we regard the problem is ill-defined and reject it.

In subsequent experiments, we report the performance of combining VCSEARCH with CoT (Wei et al., 2022) and PAL (Gao et al., 2023) to validate its effectiveness in practical applications.

Table 1: The rejection rates of various comparative methods on PMC

Method	Contra-type					Missing-type				
	Addsub	MultiArith	SVAMP	GSM8k	Avg	Addsub	MultiArith	SVAMP	GSM8k	Avg
Deepseek 6.7B										
Basic	9.83	11.97	12.48	7.97	10.56	0.54	5.75	6.06	2.92	3.82
CoT	30.73	22.28	27.24	15.68	23.98	28.99	53.97	52.06	28.34	40.84
PAL	2.86	1.94	3.62	1.96	2.59	0.27	0.00	0.84	0.79	0.48
Satlm	5.73	2.78	4.83	6.79	5.03	68.83	63.28	64.36	46.04	60.63
Ours	**54.09**	**52.64**	**54.89**	**52.67**	**53.58**	**89.70**	**88.49**	**83.51**	**63.68**	**81.35**
Qwen2.5 7B										
Basic	27.86	22.00	25.23	28.36	25.86	79.94	75.97	80.24	64.57	75.18
CoT	36.88	31.75	44.69	38.16	37.87	71.27	80.54	82.18	55.09	72.27
PAL	47.54	42.06	46.57	41.96	44.53	82.11	89.34	91.51	82.22	79,97
Satlm	12.29	9.47	16.24	23.79	15.45	74.79	62.60	66.06	44.10	61.89
Ours	**48.36**	**59.88**	**56.44**	**62.87**	**56.89**	**97.01**	**95.93**	**93.93**	**83.52**	**92.60**
Qwen2.5 3B										
Zero	29.08	23.39	34.22	28.75	28.86	47.42	54.99	71.87	54.20	57.12
CoT	34.42	36.21	42.01	30.06	35.67	63.41	73.09	80.72	51.37	67.14
PAL	3.28	7.64	5.90	11.37	7.05	17.07	10.49	26.67	17.18	17.85
Satlm	15.57	5.57	16.24	12.78	13.44	54.74	41.11	43.39	26.73	41.49
ours	**59.83**	**58.49**	**60.00**	**71.89**	**62.53**	**93.49**	**87.81**	**88.84**	**78.03**	**87.04**
Qwen2.5 1.5B										
Basic	23.36	36.49	33.15	26.92	29.98	13.00	22.50	36.72	20.72	23.23
CoT	21.72	32.59	26.30	25.35	26.49	42.27	51.60	59.63	45.17	49.67
PAL	4.91	7.52	6.04	9.80	7.06	4.06	4.74	8.48	6.83	6.03
Satlm	6.55	3.06	7.91	6.27	5.94	27.91	19.12	23.15	14.43	21.15
Ours	**38.93**	**32.59**	**43.08**	**40.91**	**38.87**	**73.44**	**63.41**	**64.48**	**47.86**	**62.29**

4 Experiments

In this section, we conduct experiments to answer the following three research questions.

RQ1. Can VCSEARCH effectively identify and reject ill-defined problems?

RQ2. Can VCSEARCH outperform formalized prompting method in modeling capabilities?

RQ3. Can VCSEARCH help existing methods achieve robust mathematical reasoning in realistic scenarios?

4.1 Experimental Setup

Datasets. We conduct experiments on two types of datasets to validate our approach and address the three research questions: ill-defined problems and well-defined problems. For **ill-defined problems**, we primarily use our proposed PMC benchmark and Mathtrap (Zhao et al., 2024) dataset, which includes mathematical trap problems (Mathtrap results in Appendix). For **well-defined problems**, we utilize the original four subsets of PMC, which is AddSub (Hosseini et al., 2014), MultiArith (Koncel-Kedziorski et al., 2016), SVAMP (Patel et al., 2021), GSM8k (Cobbe et al., 2021), as well as Robustmath (Zhou et al., 2024), where symbols serve as interference signals, and GSM-IC (Shi et al., 2023), where irrelevant information serves as interference signals.

Compared methods. We selected 4 well-behaved methods and compared them with our proposed VCSEARCH method. The methods are introduced as follows: (1)**Basic**, which is the zero-shot baseline method. (2)**CoT**, (Wei et al., 2022), let model step-by-step reasoning before providing the final answer. (3)**PAL** (Gao et al., 2023), modeling problem with python language. (4)**Satlm** (Ye et al., 2024), utilizes declarative prompting to model problems with satisfiability-aided language

Implementation Details. Our main experiments are conducted on the Qwen2.5-Coder 7B/3B/1.5B (Hui et al., 2024) and Deepseek-coder-6.7B (Guo et al., 2024). For all compared methods, we explicitly informed the model about the po-

Table 2: Comparison of the performance of Satlm and VCSEARCH on well-defined problems

Dataset	Deepseek 6.7B		Qwen 7B		Qwen 3B		Qwen 1.5B	
	Satlm	Ours	Satlm	Ours	Satlm	Ours	Satlm	Ours
Addsub	42.89	59.24	72.15	85.31	53.41	75.94	28.86	61.26
MultiArith	73.50	72.50	71.50	81.34	39.50	59.67	20.00	45.67
SVAMP	50.21	54.41	70.80	82.10	42.60	60.70	18.70	40.80
GSM8k	34.10	41.31	50.11	67.62	29.34	41.31	10.32	21.37
Robustmath	44.33	53.67	55.33	75.67	38.05	51.00	7.40	30.67
GSM-IC	18.80	24.20	49.20	74.52	22.60	39.24	5.32	12.00
Avg	43.97	50.87	61.51	77.76	37.58	54.64	15.10	35.30

tential presence of ill-defined problems. Detailed settings and prompts can be found in the Appendix.

4.2 Empirical Results

RQ1. Can VCSEARCH effectively identify and reject ill-defined problems?

Our systematic evaluation on PMC (Table 1) revealed that Contra-type tasks are more challenging than Missing-type, with all methods performing worse. VCSEARCH excelled in all-ill defined tasks, enabling all comparison models to achieve SOTA, improving the Rejection rate of identifying ill-defined problems by at least 12% across different LLMs. Further analysis showed the DeepSeek model struggled due to its tendency to preset initial values (e.g., 0) for missing data, reducing recognizability. The Qwen series performed better on ill-defined problems, but long-context prompting was highly scale-dependent. In contrast, VCSEARCH demonstrated exceptional robustness, performing consistently across models of varying sizes.

RQ2. Can VCSEARCH outperform formalized prompting method in modeling capabilities?

In this section, we systematically compare VCSEARCH with traditional few-shot prompt methods that directly utilize the SMT-Lib language as in-context (Satlm). Since the ability to solve well-defined problems is a critical criterion for evaluating the modeling capabilities of algorithms, we focus on their performance in such tasks. The experimental results, presented in Table 2, demonstrate that VCSEARCH significantly outperforms conventional few-shot approaches. This underscores the effectiveness of the decomposition and search strategies introduced in our work, particularly for smaller base models, where these strategies lead to a substantial improvement in modeling capabilities. On average, accuracy improves by 14.95%, with the most notable improvement observed in the Qwen 1.5B model, where accuracy increases from 15.10% to 35.30%. These findings show that VCSEARCH has effectively enhanced the model's ability to model problems.

RQ3. Can VCSEARCH help existing methods achieve robust mathematical reasoning in realistic scenarios?

In real-world scenarios, mathematical problems rarely fall into strictly well-defined or ill-defined categories. Instead, there is often a need to both solve well-defined problems and identify ill-defined ones. To the best of our knowledge, we are the first to explore this hybrid setting in the context of math word problems (MWP). For our experiments, we employed a balanced sampling strategy (e.g. $\mathcal{D}_w : \mathcal{D}_i = 1 : 1$) to fairly assess the ability to identify ill-posed problems and solve well-defined problems simultaneously. This evaluation strategy is analogous to how imbalanced classification studies often report balanced metrics to properly assess model performance across all classes (Thabtah et al., 2020). After three repeated experiments, we report the mean ± standard deviation in Table 3.

The results show that VCSEARCH + CoT and VCSEARCH + PAL significantly outperform traditional CoT and PAL methods in rejecting unreasonable problems. The rejection rate of ill-defined problems improved by 42.96% and 42.03% respectively, while the real-world evaluation metrics R-score gained 16.78 and 19.39 points, confirming the application value of the hybrid architecture in complex real-world scenarios. We also provide additional discussions in the appendix, including a variation of the R-score metric and experimental results under different dataset proportions.

4.3 More discussion.

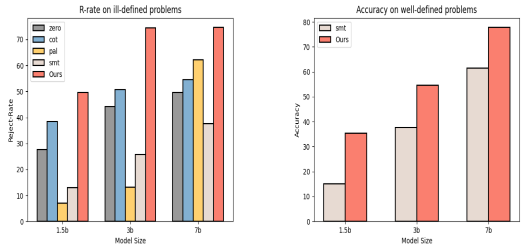
Ablations. In this part, we evaluate the impact of two core components of VCSEARCH on overall performance in Table 4. Removing the iterative search framework(just use one-time refine) results in limited improvement over the baseline

Table 3: Reaction scores of VCSEARCH + and comparison methods in a realistic environment with both ill-defined and well-defined problems

Model	Methods	Reject-Rate	R-score
Qwen2.5 3B	CoT	51.33±2.29	65.93±0.73
	+Ours	76.13±1.56	73.98±0.28
	PAL	14.46±0.41	48.56±0.22
	+Ours	75.59±1.39	74.08±1.17
Qwen2.5 1.5B	CoT	39.93±1.96	53.91±1.16
	+Ours	65.06±1.48	63.26±0.84
	PAL	7.73±2.04	32.85±1.00
	+Ours	66.66±0.24	62.28±0.65

Table 4: Ablation study on Qwen 7B model.

Search	Initialization	R-Rate	Accuracy
	✓	43.59	61.28
✓		89.97	22.81
✓	✓	74.75	77.76



(a) ill-defined problems (b) well-defined problems

Figure 4: Performance of VCSEARCH varying from different model size

SMT solver for few-shot learning. Excluding anchored initialization causes significant search space divergence, with the model becoming overly conservative and rejecting most solutions, severely impairing its ability to solve well-defined data. These findings underscore the necessity of both components.

Performance of VCSEARCH on Models of Different Sizes. Visual analysis of Qwen model results (Figure 4) reveals a strong correlation between model scale and performance: both ill-defined problem identification ability and well-defined problem solving ability decline with smaller models. However, our method mitigates this degradation and even shows advantages across scales. Specifically, VCSEARCH on Qwen-3B surpasses other methods on Qwen-7B in problem rejection and rivals SMT prompting on models an order of magnitude larger in solving well-defined problems, demonstrating its effectiveness and practical value in resource-limited scenarios.

5 Related work

Enhancing Mathematical Reasoning in LLMs Mathematical reasoning is a crucial aspect in evaluating model reasoning skills, and there are currently two predominant lines for enhancing these skills. One line involves leveraging the existing few-shot prompt tool, such as CoT (Wei et al., 2022), PAL (Gao et al., 2023). The other is centered around fine-tuning strategy, like Metamath (Yu et al., 2023), WizardMath (Luo et al., 2023) and Mugglemath (Li et al., 2023). Recent work has focused on how to achieve results that match or even exceed those of large models on smaller models (Guan et al., 2025) and smaller training datasets (Li et al., 2024a) by introducing techniques such as reinforcement learning and MCTS (Tolpin and Shimony, 2012).

Robust Mathematical Reasoning In recent years, there has been a significant surge in attention to the robustness of LLMs (Morris et al., 2020; Wang et al., 2021). In the context of robust mathematical reasoning, most existing work focuses on defining and constructing challenging "trap" datasets. For instance, Wang et.al (Wang et al., 2024) treats mathematical problems from different datasets as an out-of-distribution (OOD) generalization problem. Robustmath (Zhou et al., 2024) introduces irrelevant punctuation marks as distractors, while GSMIC (Shi et al., 2023) employs a sentence of unrelated contextual text to serve as a distractor, both aiming to investigate model performance variations. The work most similar to ours is MathTrap (Zhao et al., 2024), which focuses on a relatively small set of fewer than 300 ill-defined problems. In contrast, our PMC dataset is far more comprehensive, containing over 5,000 ill-defined problems.

6 Conclusion

This paper addresses mathematical reasoning with missing and contradictory conditions by introducing PMC, a large-scale benchmark for evaluating LLM robustness. Our observations reveal a trade-off dilemma between reasoning for well-defined problems and recognizing ill-defined problems. To solve this trade-off, we propose VCSEARCH, a training-free framework that uses formal language to detect ill-defined problems, enhanced by a variable-constraint pair search strategy to improve formal modeling. Extensive experiments show VCSEARCH achieves superior robust reasoning across diverse model architectures and sizes.

Limitations

Our work has two main limitations:

Time Consumption. Due to the use of variable-wise refinement and search architecture during the reasoning process, our method incurs higher time overhead compared to the baseline methods.

Limitations of Formal Tools. Our ability to identify ill-defined problems relies on formal tools, such as SMT solver. According to the algorithm design, the system will directly reject tasks that are unsuitable for modeling with logical tools, which may lead to the incorrect rejection of some well-defined problems.

References

Janice Ahn, Rishu Verma, Renze Lou, Di Liu, Rui Zhang, and Wenpeng Yin. 2024. Large language models for mathematical reasoning: Progresses and challenges. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics*, pages 225–237.

Clark Barrett, Aaron Stump, Cesare Tinelli, et al. 2010. The smt-lib standard: Version 2.0. In *Proceedings of the 8th international workshop on satisfiability modulo theories*, volume 13, page 14.

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

Leonardo Mendonça de Moura and Nikolaj S. Bjørner. 2008. Z3: an efficient SMT solver. In *Procddings of the 14th Tools and Algorithms for the Construction and Analysis of Systems International Conference*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning*, pages 10764–10799.

Xinyu Guan, Li Lyna Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang. 2025. rstar-math: Small llms can master math reasoning with self-evolved deep thinking. *arXiv preprint arXiv:2501.04519*.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. 2024. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

Tanmay Gupta and Aniruddha Kembhavi. 2023. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14953–14962.

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. 2014. Learning to solve arithmetic word problems with verb categorization. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 523–533.

Hui Huang, Yingqi Qu, Jing Liu, Muyun Yang, and Tiejun Zhao. 2024. An empirical study of llm-as-a-judge for llm evaluation: Fine-tuned judge models are task-specific classifiers. *arXiv preprint arXiv:2403.02839*.

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

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. Mawps: A math word problem repository. In *Proceedings of the 2016 conference of the north american chapter of the association for computational linguistics*, pages 1152–1157.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. 2022. Solving quantitative reasoning problems with language models. In *Advances in Neural Information Processing Systems*, pages 3843–3857.

Chengpeng Li, Zheng Yuan, Guanting Dong, Keming Lu, Jiancan Wu, Chuanqi Tan, Xiang Wang, and Chang Zhou. 2023. Query and response augmentation cannot help out-of-domain math reasoning generalization. *arXiv preprint arXiv:2310.05506*.

Zenan Li, Zhi Zhou, Yuan Yao, Yu-Feng Li, Chun Cao, Fan Yang, Xian Zhang, and Xiaoxing Ma. 2024a. Neuro-symbolic data generation for math reasoning. *arXiv preprint arXiv:2412.04857*.

Zenan Li, Zhi Zhou, Yuan Yao, Xian Zhang, Yu-Feng Li, Chun Cao, Fan Yang, and Xiaoxing Ma. 2024b. Neuro-symbolic data generation for math reasoning. In *Advances in Neural Information Processing Systems*.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. 2024. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*.

Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei

Lin, Shifeng Chen, and Dongmei Zhang. 2023. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. *arXiv preprint arXiv:2308.09583*.

Jingyuan Ma, Damai Dai, Lei Sha, and Zhifang Sui. 2024. Large language models are unconscious of unreasonability in math problems. *arXiv preprint arXiv:2403.19346*.

John X Morris, Eli Lifland, Jin Yong Yoo, Jake Grigsby, Di Jin, and Yanjun Qi. 2020. Textattack: A framework for adversarial attacks, data augmentation, and adversarial training in nlp. *arXiv preprint arXiv:2005.05909*.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Yang Wang. 2023. Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning. *arXiv preprint arXiv:2305.12295*.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are nlp models really able to solve simple math word problems? *arXiv preprint arXiv:2103.07191*.

Ewa Puchalska and Zbigniew Semadeni. 1987. Children's reactions to verbal arithmetical problems with missing, surplus or contradictory data. *For the learning of mathematics*, 7(3):9–16.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed Chi, Nathanael Schärli, and Denny Zhou. 2023. Large language models can be easily distracted by irrelevant context. *arXiv preprint arXiv:2302.00093*.

Fadi Thabtah, Suhel Hammoud, Firuz Kamalov, and Amanda Gonsalves. 2020. Data imbalance in classification: Experimental evaluation. *Information Sciences*, 513:429–441.

David Tolpin and Solomon Shimony. 2012. Mcts based on simple regret. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 26, pages 570–576.

Boxin Wang, Chejian Xu, Shuohang Wang, Zhe Gan, Yu Cheng, Jianfeng Gao, Ahmed Hassan Awadallah, and Bo Li. 2021. Adversarial glue: A multitask benchmark for robustness evaluation of language models. *arXiv preprint arXiv:2111.02840*.

Yiming Wang, Pei Zhang, Baosong Yang, Derek Wong, Zhuosheng Zhang, and Rui Wang. 2024. Embedding trajectory for out-of-distribution detection in mathematical reasoning. *Advances in Neural Information Processing Systems*, 37:42965–42999.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, pages 24824–24837.

Yuqing Yang, Ethan Chern, Xipeng Qiu, Graham Neubig, and Pengfei Liu. 2023. Alignment for honesty. *arXiv preprint arXiv:2312.07000*.

Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. 2024. Satlm: Satisfiability-aided language models using declarative prompting. *Advances in Neural Information Processing Systems*, 36.

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

Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Dan Zhang, Diego Rojas, Guanyu Feng, Hanlin Zhao, et al. 2024. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *arXiv preprint arXiv:2406.12793*.

Jun Zhao, Jingqi Tong, Yurong Mou, Ming Zhang, Qi Zhang, and Xuan-Jing Huang. 2024. 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*, pages 16361–16376.

Zirui Zhao, Wee Sun Lee, and David Hsu. 2023. Large language models as commonsense knowledge for large-scale task planning. In *Advances in Neural Information Processing Systems*, pages 31967–31987.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623.

Zihao Zhou, Qiufeng Wang, Mingyu Jin, Jie Yao, Jianan Ye, Wei Liu, Wei Wang, Xiaowei Huang, and Kaizhu Huang. 2024. Mathattack: Attacking large language models towards math solving ability. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19750–19758.

A Appendix

A.1 Details of PMC

We give more details of our PMC here.

A.1.1 Composition and examples of PMC

We show the number of specific subsets of PMC in Table 5, and show more representative problems to help understand our dataset.

Table 5: The specific number of rewritten datasets

Type	AddSub	MultiArith	SVAMP	GSM8k	Sum
M-type	369	591	825	1129	2914
C-type	244	359	745	765	2113

Example 1: Example 1 of PMC

Statement: Josh decides to try flipping a house. He buys a house for $80,000 and then puts in $50,000 in repairs. This increased the value of the house by 150%. How much profit did he make? # Excepted Answer: 70,000

M Version: Josh decides to try flipping a house. He buys a house for $80,000 and then puts ~~$50,000~~ some cost in repairs. This increased the value of the house by 150%. How much profit did he make?

C Version: Josh decides to try flipping a house. He buys a house for $80,000 and then puts in$50,000 in repairs. This increased the value of the house by 150%, but the market value of the house after repairs is only $100,000. How much profit did he make? (# market value Contrary to the expected)

Example 2: Example 2 of PMC

Statement: Janet's ducks lay 16 eggs per day. She eats three for breakfast every morning and bakes muffins for her friends every day with four. She sells the remainder at the farmers' market daily for $2 per fresh duck egg. How much in dollars does she make every day at the farmers' market? # Excepted Answer: 14

M Version: Janet's ducks lay 16 eggs per day. She eats ~~three~~ some for breakfast every morning and bakes muffins for her friends every day with four. She sells the remainder at the farmers' market daily for $2 per fresh duck egg. How much in dollars does she make every day at the farmers' market?

C Version: Janet's ducks lay 16 eggs per day. She eats three for breakfast every morning and bakes muffins for her friends every day with four. She sells the remainder at the farmers' market daily for $2 per fresh duck egg. How much in dollars does she make every day at the farmers' marketif she give 10 eggs away to her neighbor? (# She only left 9 eggs, can not give away 10 eggs)

A.1.2 Constrction prompt

The construction prompt we used is shown in the example 3,4,5.

Example 3: Constrction prompt for missing type

Given the following math problem, identify all the variables and constraints involved. Then, modify the problem by replacing a key numerical value in one of the constraints with an indefinite placeholder (e.g., "some number", "a certain value", etc.), such that the resulting problem lacks sufficient information to determine a unique solution.
You can answer with following step:
Step 1: Variable and Constraint Identification.
Step 2: Decide the mutated Variable or constraint and explain the reason.
Step 3: Answer with final mutated problem.
Original Problem: {Problem}
Modified Problem: [Your answer]

Example 4: Constrction prompt for contra type

Given the following math problem, identify all the variables and constraints involved. Then, modify the problem by introducing an additional constraint that directly conflicts with an existing one. The resulting problem should contain contradictory information that makes it logically unsolvable.
You can answer with following step:
Step 1: Variable and Constraint Identification.
Step 2: Decide the mutated Variable or constraint and explain the reason.
Step 3: Answer with final mutated problem.
Original Problem: {Problem}
Modified Problem: [Your answer]

Example 5: Validation prompt

Given the following math problem, determine whether it is solvable. If not, identify why the problem is ill-defined. Specifically, analyze whether the conditions provided are insufficient or self-contradictory, making it impossible to derive a unique solution.
You can answer with the following steps:
Step 1: Variable and Constraint Identification.
Step 2: Analyze whether the problem is solvable under the given constraints. If it is unsolvable, explain whether it is due to missing information or contradictory conditions, and identify the responsible part(s).
Step 3: Give the final feedback if the question is unsolvable
Problem: {Problem}
Answer: [Your answer]

A.1.3 Human annotators

When the LLM used for verification outputs inconsistent responses, we will enable human annotators to verify. Our annotators come from within the lab, no more than 5 master's and doctoral students.

A.2 Details of VCSEARCH

In this part we will introduce the details in our algorithm.

A.2.1 Prompts in VCSEARCH

We show the prompts we use in VCSEARCH with examples 6 and 7.

A.2.2 Formal tools

The SMT-LIB(Satisfiability Modulo Theories Library) (Barrett et al., 2010) is a tool for working with satisfiability problems. It provides a standard notation compatible input language for representing logical formulas. And powful SMT solvers, such as Z3 (de Moura and Bjørner, 2008), extend the classical boolean satisfiability problem (SAT problem) to enable verification of numerical arithmetic problems, among others. The SMT solver will initially determine whether the modeled problem is satisfiable (SAT/UNSAT). If it is satisfiable, the solver will then provide a feasible solution within the feasible domain of the problem. Specifically, we use z3 as a formal tool in the paper.

A.2.3 Double-check solving strategy with SMT solver

We use a double-check strategy when checking with the SMT solver. Specifically, we verify both the satisfiability of the formal expression and the uniqueness of the solution. To be specific, to check the satisfiability of the formal expression, we utilize the Z3 solver. This strategy regards the problem as ill-defined and rejects the answer if the formal expression is unsatisfiable(UNSAT). To assess the uniqueness of the solution, We develop this check through a two-stage process. First, we utilize the Z3 solver to determine one solution and subsequently incorporate this candidate solution as a constraint into the formal expression. If the formal expression remains satisfiable, then it implies that the formal expression encompasses multiple solutions, leading the strategy to reject the answer as it violates the uniqueness of the answer.

To be precise, in the solution phase, our strategy let the SMT solver return four possible different values:

- **Error**: Indicates that the modeling cannot be successfully completed. Similar to a compilation error, we do not consider it as a valid state.
- **UNSAT**: Indicates that the modeling state cannot be satisfied, there are contradictory conditions, and the answer is rejected.
- **Multi**: We believe that the question is ambiguous, resulting in multiple solutions, and the answer is rejected.
- **Ans**: Returns a normal real number, representing the answer to the question.

A.2.4 A example for VCSEARCH

Our approach to determining variable-constraint relationships is as follows:

- **Preparation Phase (Variables → Constraints)**: For a given variable, directly retrieve all constraints containing that variable from the constraint pool.
- **Update Phase (Constraints → Variables)**: For a given constraint, we identify all new associated variables in it.

To further illustrate this method, we present a concrete example using a contra-type problem in PMC (example 8) to demonstrate the search process:

Example 6: prompts used in VCSEARCH-1

Refine module prompt

I have previously asked you to write Z3 constraints for a problem. However, the current set of constraints for the variable may have omissions or errors. I would like you to review it from the following two aspects and make appropriate modifications if necessary:
1. Based on the problem description, consider whether the current constraints accurately capture the problem.
2. Add constraints based on real-world knowledge, considering whether there are any missing modeling statements, such as the quantity of items should be >= 0, or the relationships between the sides of a triangle.
Please note that you only need to add constraints to the CURRENT HEAD VARIABLE; in other words, the new constraints MUST include the head variable!
You can first provide your thought process, and then write the new constraints that include the head variable after the identifier <SOS>

You can follow the example:

Question: Josh decides to try flipping a house. He buys a house for $80,000 and then puts in $50,000 in repairs. This increased the value of the house by 150%, but the market value of the house after repairs is only $100,000. How much profit did he make?

Existing Constraints:

solver.add(initial-cost == 80000)

solver.add(total-investment == initial-cost + repair-cost)

solver.add(repair-cost == 50000)

solver.add(increased-value-percentage == 0.5) # 150% increase

solver.add(expected-value == initial-cost * (100 + increased-value-percentage))

solver.add(market-value-after-repairs == 100000)

solver.add(total-investment >= 0)

solver.add(profit == market-value-after-repairs - total-investment)

solver.add(profit >= 0)

solver.add(expected-value >= 0)

Now head variable: expected-value

Now existing constraints with head variable:

solver.add(expected-value == initial-cost * (100 + increased-value-percentage))

solver.add(expected-value >= 0)

Answer:

1. For the constraint expected-value == initial-cost * (100 + increased-value-percentage), the equation for expected-value in the problem should be initial-cost * (1 + increased-value-percentage). Therefore, this constraint should be modified to solver.add(expected-value == initial-cost * (1 + increased-value-percentage)).

2. For the constraint solver.add(expected-value >= 0) aligns with real-world requirements. Additionally, since expected-value is an unknown variable, it is appropriate to add real-world constraints, so this should be retained.

3. Furthermore, expected-value and market-value-after-repairs refer to the same entity in the problem, so a constraint should be added: market-value-after-repairs == expected-value.

<SOS>

So, new Constraints with head variable is

solver.add(expected-value == initial-cost * (1 + increased-value-percentage))

solver.add(expected-value >= 0)

solver.add(expected-value == market-value-after-repairs)

Question:{question}

Existing Constraints:{constraint}

Now head variable:{head}

Now existing constraints with head variable:{constrain-head}

Answer:

Example 7: prompts used in VCSEARCH-2

Verification module prompt
Please judge which set of constraints is better for the given problem, including all constraints of variable "X".
Problem: {question}
variable:{head}
Constrains set1:{cons1}
Constrains set1 ans:{cans1}
Constrains set2:{cons2}
Constrains set2 ans:{cans1}
Please write down your thinking process first, and finally output, "I think Constrains set1 is better", or "I think Constrains set2 is better".

A.3 Details of Experiment

A.3.1 Setup

Compared methods. We selected three representative few-shot prompting methods, along with the zero-shot method that utilizes the intrinsic capabilities of the model, and compared them with our proposed VCSEARCH method. The methods are introduced as follows: (1)**Basic**, which is the zero-shot baseline method, directly feeds the problem and instructions to the LLMs without any example problem in the context. (2)**CoT**, (Wei et al., 2022), requires the model to explicitly output intermediate step-by-step reasoning through natural language before providing the final answer. (3)**PAL** (Gao et al., 2023), converts each step of problem-solving into a programming language format and subsequently utilizes an external programming language interpreter for execution, thereby obtaining the results. (4)**Satlm** (Ye et al., 2024), utilizes SMT-LIB to model the problems, then uses an external SMT solver to check for a feasible solution to the problem as well as obtain the ground-truth answer.

Prompts. For the few-shot prompting methods, we prepared four contextual examples (4-shot) for each method, consisting of two well-defined problems and two ill-defined problems. In the system prompt, we explicitly informed the model about the potential presence of ill-defined problems. If the model determines that a problem is unsolvable, it is instructed to output a statement containing the term "unsolvable." This allows us to evaluate whether the model successfully identifies ill-defined problems.

Set up details for Sec4.3. At this part, we employed a balanced sampling strategy to fairly assess the ability to identify ill-posed problems and solve well-defined problems simultaneously. (with a solvable/unsolvable problem ratio of $\alpha = 1 : 1$), selecting 500 samples from the ill-defined problem set (Table 1) and the well-defined problem set (Table 2) to construct a 1000-sample test set. After three repeated experiments, we report the mean ± standard deviation in Table 3.

A.3.2 Prompts used in Preliminary experiments

We show the prompts we use in preliminary experiments to reflect the trade-off dilemma with examples 9.

A.3.3 More experiment results

Table 6: R-Rate on MathTrap

Model	Deepcoder	Qwen7b	Qwen3b	Qwen1.5b
Zero	22.95	15.57	15.57	13.72
Ours	65.57	86.06	88.89	74.59

Here, we also tested our method on several other benchmarks that involve refusal to answer. Our method also demonstrated superior performance on MathTrap. However, MathTrap's mathematical problems

Example 8: Example in VCSEARCH

"Josh decides to try flipping a house. He buys a house for 80,000 ***and then puts in*** *50,000 in repairs. This increased the value of the house by 150%, but the market value of the house is only $100,000. How much profit did he make?"*

After the initialization step, we obtain an initial constraint system, represented in Python Z3 code. This system consists of a variable queue and a constraint pool.

Variables:

```
"initial-cost", "repair-cost", "increased-value-percentage",
"expected-value", "market-value-after-repairs", "profit",
"total-investment"
```

Constraints:

```
initial-cost == 80000
repair-cost == 50000
market-value-after-repairs == 100000
increased-value-percentage == 0.5
total-investment == initial-cost + repair-cost
expected-value == initial-cost * (100 + increased-value-percentage)
profit == market-value-after-repairs - total-investment
```

After the Initialization, assume that the first element in the variable queue is "expected-value", we will demonstrate a single iteration of the search process.

Preparation

Identify constraints involving this variable "expected-value":

```
expected-value == initial-cost * (100 + increased-value-percentage)
```

Exploration

Utilize LLM knowledge to refine the constraints by generating a constraints set with the head variable "expected-value":

```
expected-value == initial-cost * (basic_multiplier + increased-value-percentage)
basic_multiplier == 1
```

Verification

Compare the original constraint system with the refined one and select the better version.
(In this case, the newly generated constraint set is selected).

Update

Replace the outdated constraint with the refined one.
Identify any newly introduced variables (e.g., "basic_multiplier") and append them to the tail of the variable queue for subsequent iterations.

Example 9: prompts used in Preliminary experiments

Pure prompt for ill-defined problem
Now we have some math problems that may be ill-defined. Please judge whether they are indeed ill-defined (no unique real number solution can be determined). If there is indeed no solution, answer true, otherwise answer false. Explain the reason first and then answer.

Pure prompt for well-behaved problem
You're an experienced elementary school teacher, and I'm now expecting you to solve some math problems.

Mixed prompts
You're an experienced elementary school teacher, and I'm now expecting you to solve some math problems. If you find these problems unsolvable, please output "this is unsolvable". Or please solve this answer, and give the final answer with format "The answer is X"

involve a significant amount of geometry and algebra, which are not well-suited for formal tool modeling. This is also not suitable for methods such as PAL. So we only compare ours with zero-shot method. In such scenarios, our method adopts a relatively conservative approach, rejecting any problem it cannot confidently solve in order to maintain the safety of the reasoning system.

A.3.4 Discussion about reasoning in realistic scenarios

Discussion of dataset ratios

In our paper, we adopted a balanced setting($i.e.$, $D_w : D_i = 1 : 1$) to measure the reaction score. This balanced approach allows us to evaluate the capability of methods to both answer well-defined problems and reject ill-defined problems with equal importance. This evaluation strategy is analogous to how imbalanced classification studies often report balanced metrics to properly assess model performance across all classes (Thabtah et al., 2020). By maintaining this balanced setting, we provide a more comprehensive and fair assessment of each method's capabilities of answering and rejecting. Additionally, we compared the R-score performance across different dataset ratios (defined as $\alpha = D_w : D_i$) on the Qwen1.5B model, and our method consistently demonstrated superior results.

Table 7: Performance among different data ratios

α	0.2	0.5	1	2	5
CoT	44.61 ± 1.02	49.58 ± 2.00	53.91 ± 1.16	58.96 ± 0.78	62.83 ± 1.55
CoT + Ours	64.40 ± 0.43	64.05 ± 0.60	63.26 ± 0.84	64.33 ± 0.89	62.91 ± 0.79
PAL	16.01 ± 0.66	24.03 ± 1.12	32.85 ± 1.00	41.15 ± 0.49	49.53 ± 2.89
PAL + Ours	65.26 ± 1.54	62.46 ± 0.22	62.28 ± 0.65	58.55 ± 1.13	58.84 ± 0.56

More convincing metrics

To prevent excessive score inflation through question rejection (where rejecting all questions would yield only 50% of the total score), we introduce the R*-score metric as below

$$\frac{\sum_{p \in D_i} \mathbb{I}[f(p) = \text{Reject}] + \sum_{p \in D_w} \mathbb{I}[f(p) = g(p)]}{|D_i| + |D_w|}$$

◁ ▷

We evaluate our method under balanced settings and present the results in the following table. Our approach maintains superior performance in most scenarios(R*-score), demonstrating that our performance gains do not stem from simply rejecting most questions.

Table 8: Perfomance among R-score and R*-score

	Qwen 1.5B		Qwen 3B	
Method	R-score	R*-score	R-score	R*-score
CoT	53.91 ± 1.16	51.10 ± 2.08	65.93 ± 0.73	65.10 ± 1.04
CoT + Ours	63.26 ± 0.84	53.10 ± 0.06	73.98 ± 0.28	66.93 ± 0.28
PAL	32.85 ± 1.00	30.63 ± 0.18	48.56 ± 0.22	47.66 ± 0.49
PAL + Ours	62.28 ± 0.65	51.90 ± 1.15	74.08 ± 1.17	65.73 ± 1.30